Efficient Data Retrieval for Large-Scale Smart City Applications through Applied Bayesian Inference

Jin Ming Koh*, Marcus Sak*,
Hwee-Xian Tan†, Huiguang Liang†, Fachmin Folianto† and Tony Quek‡
*NUS High School of Mathematics and Science, Singapore
†Institute for Infocomm Research (I^{2}R), Singapore
‡Singapore University of Technology and Design (SUTD), Singapore

***Abstract*—Recent years have witnessed the proliferation of worldwide efforts towards developing technologies for enabling smart cities, to improve the quality of life for citizens. These smart city solutions are typically deployed across large spatial regions over long time scales, generating massive volumes of data. An efficient way of data retrieval is thus required, for post-processing of the data - such as for analytical and visualization purposes.**

In this paper, we propose a data prefetching and caching algorithm based on Bayesian inference, for the retrieval of data in large-scale smart city applications. A brute-force approach is used to determine the optimal weight correction factor in the proposed prefetching algorithm. We evaluate the optimized Bayesian prefetching algorithm against the Naïve and Random prefetch baselines, using both simulated and actual data usage patterns. Results show that the Bayesian approach can achieve up to 48.4% reductions in actual user-perceived application delays during data retrieval.

I. Introduction

By the year 2050, 70% of the world's population is expected to live in urban and surrounding areas [1]. The resulting high population densities and continued increases in life quality expectations will necessitate cities to integrate technologies into their functional frameworks [2]. Subsequently, there has been rapid proliferation of worldwide efforts in recent years, towards developing technologies for enabling smart cities, to improve the quality of life for citizens.

Smart city solutions are technologically predicated on the emerging Internet of Things (IoT), which comprise networks of nodes that interact with the physical world, and provide services for information transfer, analytics as well as communications [3]. Access to sensory input (e.g. noise levels, humidity etc.) will increase transparency of citizens' urban lifestyle, leading to higher standards of living. For instance, government agencies can deduce sources of air pollution by following the directional increase in air particulate levels.

These solutions are typically deployed across large spatial regions over long time scales, generating massive volumes of big data of varying velocities, varieties, veracities and values. Efficient methods for data storage and retrieval are thus required to manage the data, and to provide acceptable levels of Quality of Service (QoS) to the users of the data.

In this work, we focus on the efficient retrieval of data that is generated by large-scale smart city applications, for the class of time-sensitive use cases. Examples of such use cases include the visualization of real-time data across different spatial locations, and the retrieval of node-specific information for network management purposes. While the timeliness of the data retrieval in these use cases is not mission-critical, the delay incurred during the data retrieval process provides varying QoS levels to the end users.

Based on literature on prefetching and caching, we propose an algorithm based on Bayesian inference, to minimize the delay incurred for data retrieval. Prior work by Cao et al [4] has demonstrated that the performance of the combination of both prefetching [5] [6] [7] and caching [8] is better than any existing singular approach, and can potentially increase performance by 50%. In addition, such an approach is complementary to data retrieval optimization techniques that may be implemented at the backend systems - such as optimizing SQL queries and database designs.

A brute-force approach is used to determine the optimal weight correction factor in the proposed prefetching algorithm. We incorporate the proposed algorithm into *Sensorem*, a mobile visualization platform for wireless sensor networks that we have developed [9]. We evaluate the optimized Bayesian prefetching algorithm against the Naïve and Random prefetch baselines, using both simulated and actual data usage patterns on *Sensorem*. Results show that the Bayesian approach can achieve up to 48.4% reductions in actual user-perceived delays during data retrieval.

The rest of the paper is structured as follows. Section II provides the background on prefetching and caching mechanisms, as well as descriptions of the baseline prefetch policies. We present our proposed Bayesian-based prefetch algorithm in Section III. We optimise and evaluate our algorithm through a simulation-based brute-force approach in Section IV. In Section V, we present the results and analysis of our experiments using actual data usage patterns. We highlight key conclusions and future work in Section VI.

II. Preliminaries

A. Prefetching and Caching - An Overview

Generally, prefetching and caching mechanisms comprise two stages [4]:

Algorithm 1 Random Prefetch Algorithm

```
1: index $i$ = rand[1,$n_s$]
2: prefetch($i$)
```

1) prediction of the data that the user will subsequently request for; and
2) fetching of the data from persistent remote storage (such as a backend server), and storing the data in the local cache.

When prefetched data matches subsequent user requests, the need to retrieve data from remote storage is eliminated, thereby alleviating CPU and network load. Subsequently, the overall user-perceived delay is greatly reduced, leading to improved data retrieval performance and user experience [10]. The effectiveness of the prefetch policy can thus be quantified by its *hit rate* - the probability that the next requested data is already prefetched and cached.

B. Baseline Prefetch Policies

We introduce two baseline prefetch policies, viz. Random Prefetch (RP) and Naïve Prefetch (NP), that can be used to benchmark the performance of other prefetch algorithms. The Random Prefetch policy is based on the assumption that there is no particular order and/or preference in which users request for data. The Naïve Prefetch policy assumes that data usage patterns exist for an arbitrary user of the system.

In the context of large-scale smart city applications, *data* typically refers to information that is acquired about each of the node that has been deployed. Such data can be classified into: (i) application-specific data (such as air-quality and noise data from sensor nodes that are deployed in an environmental monitoring application); and (ii) network management data (such as the battery level of each sensor node).

We consider a smart city application comprising the deployment of n_s of such sensor nodes - each with a unique identifier from 1 to n_s. Then, the Random Prefetch algorithm (Algorithm 1) will continuously prefetch data from a uniform-randomly selected node out of all the n_s nodes.

The Naïve Prefetch policy is built upon the assumption that each class of user will request for data in a sequence that is dependent on a particular property of a data modality. For instance, in an air quality monitoring application, a citizen may be interested in obtaining more detailed information (such as time-series data) of only the nodes that indicate that a particular region is having unhealthy air pollution levels. On the other hand, the network management personnel of the same air quality monitoring application may only be interested in nodes that indicate low battery levels (and thus requiring some immediate maintenance attention) or outlier air quality readings (which may be an indication of hardware issues).

Without any loss in generality, we consider a particular data property, accessed at time instance i, which we notate as $\mathcal{P}_i$, with binary classification values $\mathbf{B}_0$ and $\mathbf{B}_1$, i.e. $\mathcal{P}_i = \{\mathbf{B}_0, \mathbf{B}_1\}$. For example, data property $\mathcal{P}_i$ can refer to the battery level of each node, which can be classified as low ($\mathbf{B}_0$) or high ($\mathbf{B}_1$). Two properties, $\mathcal{P}_i$ and $\mathcal{P}_j$, accessed at different time instances i and j, are defined to be equivalent if $\mathcal{P}_i = \mathbf{B}_0$ and $\mathcal{P}_j = \mathbf{B}_0$, Similarly, $\mathcal{P}_i = \mathcal{P}_j$ if $\mathcal{P}_i = \mathbf{B}_1$ and $\mathcal{P}_j = \mathbf{B}_1$. They are not equal otherwise.

Algorithm 2 Naïve Prefetch Algorithm

```
1: $n_0$ = number of nodes with property $\mathbf{B}_0$
2: $n_1$ = number of nodes with property $\mathbf{B}_1$
3: if $n_0 > n_1$ then
4:     $\mathbf{B}_T = \mathbf{B}_0$
5: else
6:     $\mathbf{B}_T = \mathbf{B}_1$
7: end if
8: index $j$ = node with property $\mathbf{B}_T$ and smallest distance
   from the accessed node at latest time instance $i$
9: prefetch($j$)
```

We let n_0 and n_1 represent the number of nodes that are exhibiting properties $\mathbf{B}_0$ and $\mathbf{B}_1$ respectively, where $n_s = n_0 + n_1$. The data property value that is predicted to be accessed next, $\mathbf{B}_T$, is defined (for this Naïve Policy) as the property value that has been exhibited by most of the nodes. Data from the node with the smallest *distance* from the last accessed node is then prefetched, where *distance* can refer to the geographical distance or distance in a visual representation of the nodes. The Naïve Prefetch policy is summarized in Algorithm 2.

III. Proposed Bayesian Prefetch Algorithm

We now propose a Bayesian prefetch algorithm that considers both the short term and long term usage patterns of an arbitrary user, to better predict subsequent data retrieval requests.

A. Brief Primer

We first define the Request Sequence RS as a list of ***all*** accessed data property values, by that user, in chronological order. For tractability, once the type of data property under consideration is fixed (e.g. battery level), all the properties within the RS will be of that type.

For a time-series of accessed data property values, starting from the earliest to the latest accessed data property at the latest time instance t, RS is therefore an ordered, *strictly-continuous*, set of $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{t-1}, \mathcal{P}_t)$. We then consider a subsequence $S \in RS$ of the request sequence, of arbitrary length $|S|$ where subsequences of $(\mathcal{P}_i, \mathcal{P}_{i+1}, \ldots, \mathcal{P}_{i+\alpha})$ and $(\mathcal{P}_j, \mathcal{P}_{j+1}, \ldots, \mathcal{P}_{j+\alpha})$ are considered to be equivalent if the individual properties are equivalent. For example, a subsequence $(\mathcal{P}_i = \mathbf{B}_0, \mathcal{P}_{i+1} = \mathbf{B}_1, \mathcal{P}_{i+2} = \mathbf{B}_0)$ is equal to another subsequence $(\mathcal{P}_j = \mathbf{B}_0, \mathcal{P}_{j+1} = \mathbf{B}_1, \mathcal{P}_{j+2} = \mathbf{B}_0)$.

The number of data accesses in S with property $\mathbf{B}_m$ for $m \in \{0, 1\}$ is denoted by $n_m[S]$. Based on Bayes' Theorem, the probability $\mathbb{P}(\mathbf{B}_m|S)$ that data of property $\mathbf{B}_m$ will be next accessed, given a subsequence $S = (\mathcal{P}_i, \mathcal{P}_{i+1}, \ldots, \mathcal{P}_{i+\alpha})$, for any $i \in \{1, \ldots, |S|\}$, $\alpha \in \{0, \ldots, |S| - i\}$, is given by:

$$\mathbb{P}(\mathbf{B}_m|S) = \frac{\mathbb{P}(S|\mathbf{B}_m) \cdot \mathbb{P}(\mathbf{B}_m)}{\mathbb{P}(S)}, \quad (1)$$

where:

$$\mathbb{P}(\mathbf{B}_m) = \frac{n_m[S]}{|S|}; \tag{2}$$

$$\mathbb{P}(S) = \frac{\text{total no. of } S \text{ in } RS}{\text{total no. of subsequences of length } |S|}; \tag{3}$$

$$\mathbb{P}(S|\mathbf{B}_m) = \frac{\text{no. of } S \text{ proceeded by } \mathbf{B}_m \text{ in } RS}{\text{total no. of } S \text{ in } RS}. \tag{4}$$

B. Bayesian Prefetch Algorithm

Now, we define the subscripted subsequence S_k to be the subsequence $\big(\mathcal{P}_k, \mathcal{P}_{k+1}, \ldots, \mathcal{P}_{|RS|-1}, \mathcal{P}_{|RS|}\big)$, where $k \in \{1, \ldots, |RS|\}$, and $|S_k| = |RS| - k$.

Then, we compute the weighted sum, ϕ_m, of the probability $\mathbb{P}(\mathbf{B}_m|S_k)$ for each data property value $\mathbf{B}_m$, across all the possible subsequences S_k of RS, as follows:

$$\phi_m = \sum_{k=1}^{|RS|} w_{S_k} \cdot \mathbb{P}(\mathbf{B}_m|S_k), \tag{5}$$

where w_{S_k} is a weight that is assigned to each subsequence $S_k \in RS$. $\mathbb{P}(\mathbf{B}_m|S_k)$ can be evaluated using Equation (1). The value to be assigned to each weight w_{S_k} will be discussed in Section III-C.

To recapitulate, the goal of the prefetch algorithm is to identify the next likely value, $\mathbf{B}_T$, of the property $\mathcal{P}_{|RS|+1}$ that the user will fetch. The proposed Bayesian prefetch algorithm here is therefore:

$$\mathbf{B}_T = \underset{m}{\text{argmax}}\ \phi_m. \tag{6}$$

C. Dynamic Weight Adaptation

The weight w_{S_k} enables sample subsequences of different lengths to have varying importance during the data prediction phase. This is important as each data user is likely to exhibit different usage patterns. For instance, better predictions can be achieved if short-term usage patterns are allocated higher importance than long-term usage patterns, for a user who accesses data in a haphazard fashion. Conversely, users who assess data with consistent usage patterns (such as frequently requesting for data from nodes with a particular data property) will experience better prediction accuracies if their long-term usage patterns are given more importance.

However, it is typically difficult to know the usage pattern of a user *a priori*. Hence, we propose and incorporate a dynamic weight adaptation algorithm into our prefetching algorithm, based on the real-time feedback of the prediction accuracy of the system.

The weight adaptation algorithm works as follows: Initially, the weight of each subsequence S_k is assigned a value of one, i.e., $w_{S_k} = 1\ \forall\ S_k \in RS$. Suppose the current iteration of prefetching yields: (i) a correct prediction of the next accessed data property $\mathcal{P}_{|RS|+1}$ after the current request sequence RS, for an arbitrary subsequence S_{k_1}; and (ii) an incorrect prediction for another arbitrary subsequence $S_{k_2} \neq S_{k_1}$. Then, the weights $w_{S_{k_1}}$ and $w_{S_{k_2}}$ corresponding to subsequence S_{k_1} and S_{k_2} will be updated as follows:

$$w_{S_{k_1}} = w_{S_{k_1}} + w_c;\ \text{and} \tag{7}$$

$$w_{S_{k_2}} = \max(0, w_{S_{k_2}} - w_c). \tag{8}$$

Algorithm 3 Updating weights given the last accessed property, $\mathcal{P}_{|RS|}$.

```
1: Input: Set of subsequences $\mathbf{S}_{|RS|-1} = \big(S_1, S_2, .., S_{|RS|-1}\big)$; set of weights $\mathbf{w}_{|RS|-1} = \big(w_{S_1}, w_{S_2}, \ldots, w_{S_{|RS|-1}}\big)$ for each subsequence $S_k$, where $k \in \{1, \ldots, |RS|-1\}$; last predicted data property $\hat{\mathcal{P}}_{|RS|}$; and last accessed data property $\mathcal{P}_{|RS|}$.
2: Output: Updated set of weights, $\mathbf{w}_{|RS|}$.
3: for $k = 1$ to $|RS| - 1$ do
4:     if $\hat{\mathcal{P}}_k = \mathcal{P}_{|RS|}$ then
5:         $w_{S_k} = w_{S_k} + w_c$
6:     else
7:         $w_{S_k} = \max(0, w_{S_k} - w_c)$
8:     end if
9: end for
10: $\mathbf{w}_{|RS|} = \mathbf{w}_{|RS|-1} \bigcup w_{S_{|RS|}}$, where $w_{S_{|RS|}} = 1$
```

This dynamic adaptation of weights allows the Bayesian-based prefetch algorithm to adapt to different user styles over time as well as varying user behavior even for the same user. The dynamic weight adaptation procedure is summarized in Algorithm 3.

We note that there must be an optimal correction factor, denoted w_c^o that yields the most accurate long-term predictions. An excessively small w_c under-compensates for user style and places inappropriate emphasis of subsequences of different lengths, while an excessively large w_c overcompensates. We derive the optimal value of w_c^o in Section IV.

An important aspect of the proposed Bayesian algorithm is that the number of prediction parameters can easily be extended to include multiple data properties. The mathematical procedures are fully scalable, thereby supporting future generalizations and development.

The time complexity of the algorithm is estimated to be $\mathcal{O}(n^2 m)$, where n is the total size of considered RS, and m, the number of prediction parameters. Realistically, values of n and m are expected to be small. This indicates excellent expected performance under realistic conditions.

D. Data Retrieval Parallelism

We note that the amount of data that can be prefetched in each prefetch cycle is dependent on the amount of time that the prefetch iteration takes. The length of each prefetch cycle is given by $T_{cycle} = T_{cpu} + T_s + T_c$, where T_{cpu} denotes the time required to compute a suitable prefetch target, T_s denotes the time required to fetch data from the remote storage, and T_c denotes the time required to cache the data.

The maximum units of data that can be prefetched between user requests can be computed by:

$$n_{cycle} \cdot \lfloor \frac{T_u}{T_{cpu} + T_s + T_c} \rfloor, \tag{9}$$

where n_{cycle} represents the number of units of data that are fetched in parallel in each prefetch cycle; and T_u represents the time between user requests.

The intricacy lies in the fact that prefetching itself increases network traffic, and therefore contributes to the backend bottlenecks. Excessively high prefetch rates may exacerbate the backend bottlenecking. To avoid this, we use $n_{cycle} = 1$ throughout our work.

E. Cache Policy

Due to the limited size of the cache memory, a cache replacement policy is required to purge prefetched data from the cache when it is full. One important aspect of the cache policy is *information freshness*, i.e., data that has been prefetched but not used immediately may become stale.

In our work, we propose and use a Least Recently Used cache replacement policy with Early Drop (LRU-ED). In this scheme, data that has been least recently used will be purged from the cache when it is full. In addition, a cache replacement procedure is performed before each data prefetch to remove data that has exceeded a freshness threshold. This enforces the system to re-fetch updated data from the server upon user request and/or the prefetch algorithm.

IV. Optimization and Evaluation of Bayesian Prefetching through Simulations

In Section IV-A, we first discuss user behavior characteristics and heuristics, and introduce an algorithm to generate request sequences for evaluation purposes. We then determine the optimal correction factor w_c^o in the Bayesian prefetch algorithm in Section IV-B. We evaluate the performance of the Bayesian prefetch against the Naïve and Random prefetch baselines, through simulations, in Section IV-C.

A. RS Generation based on User Behaviour Characterization

We seek to model the usage pattern of our target user population, such that large numbers of Request Sequences RS can be generated for optimisation of the prefetching algorithms. We make two observations of data access patterns of actual users, based on logs of their data usage history. It should be noted that this method of characterizing user behaviour can be extended to include any number of behavioural parameters and is therefore sufficiently flexible to model usage patterns beyond our target user group.

1) Close-proximity data: Users tend to access data from nodes that are displayed in close-proximity of previously accessed nodes. Such an access pattern can be characterized by the jump number parameter n_j, which is proportional to the *display proximity* between the last accessed node data and the next requested node data. For instance, we consider a set of nodes $V = \{v_1, v_2, v_3\}$, where $d_{i,j}$ denotes the 'distance' between two nodes v_i and v_j. Here, 'distance' can take on various definitions - such as the Euclidean distance between nodes in a geographic map, or the list order of nodes in a list view. Let v_1 be the last accessed node, and $d_{1,2} < d_{1,3}$. Then, v_2 has jump number $n_j = 1$, and v_3 has jump number $n_j = 2$.

2) Similar data properties: Users tend to access data from nodes that have similar data properties as previously accessed nodes. This is characterized by the repeat number parameter n_r, which refers to the number of consecutively accessed nodes with the same data property. For instance, network management personnel may consistently request for data only from nodes with low battery levels, while a concerned citizen may consistently request data only from nodes with high noise levels or air pollution levels.

Algorithm 4 RS generation algorithm

```
1: Input: probability density functions $f_{n_r}$ and $f_{n_j}$; bias $Q$; ordered set of available nodes $V$; required request sequence length $L$
2: Output: generated request sequence $GS$; and corresponding generated node access list $N$
3: $GS = \emptyset$; $N = \emptyset$
4: while $|GS| < L$ do
5:     $a = \text{rand}[0,1]$; $b = \text{rand}[0,1]$; $c = \text{rand}[0,1]$
6:     $n_r = F_{n_r}^{-1}(a)$
7:     $n_j = F_{n_j}^{-1}(b)$
8:     $\gamma = \min(n_r, L - |GS|)$
9:     while $\gamma > 0$ do
10:        index $i = \text{rand}[1, |V|]$
11:        if $c < Q$ then
12:            $GS = GS \bigcup \mathbf{B}_0$
13:        else
14:            $GS = GS \bigcup \mathbf{B}_1$
15:        end if
16:        $N = N \bigcup V_i$.
17:        $V = V \setminus V_i$
18:        $\gamma = \gamma - 1$
19:    end while
20: end while
```

The frequency distributions of the jump number n_j and repeat number n_r for our target user population are determined through statistical analysis of data usage logged from 80 user request sequences, each comprising at least 50 nodes. Figures 1 and 2 illustrate the frequency distribution and the best-fit probability density function (pdf) for the jump number n_j and repeat number n_r respectively.

These pdfs of the jump number n_j and repeat number n_r are incorporated into the RS generation algorithm as shown in Algorithm 4, together with a bias parameter Q. The bias is defined as the proportion of elements in the RS that exhibit the data property $\mathbf{B}_0$. The set of available nodes in the network is denoted as V, and we refer to the k^{th} element in V as V_k. The generated request sequence $GS = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{L-1}, \mathcal{P}_L)$ provides a chronological list of data properties of length L, where $\mathcal{P}_i = \{\mathbf{B}_0, \mathbf{B}_1\}$ as earlier defined. The corresponding generated node access list is $N = \{N_1, N_2, ..., N_L\}$ is an ordered list of nodes that will be accessed, such that node N_i has data property $\mathcal{P}_i \in GS$. Both the generated GS and N will be used as inputs for simulation evaluation.

B. Determining the Optimal Weight Correction Factor w_c^o

We determine the optimal weight correction factor w_c^o of the Bayesian prefetch algorithm, through a brute-force simulation approach. Each *simulation cycle* is supplied with a set of 40 generated request sequences GS comprising 200 nodes per GS. Each simulation cycle is then run for $0 \leq w_c \leq 1$ with step size 0.05 and bias $Q = 0.76$. This value of bias is obtained by computing the statistical average for the target user group.

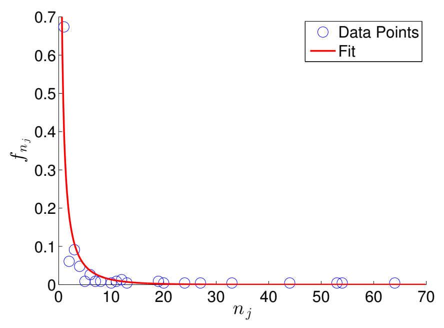

Fig. 1. Plot of probability density function for jump number n_j. The distribution demonstrates user preference towards close-proximity data.

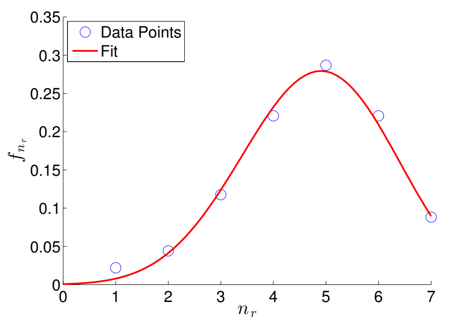

Fig. 2. Plot of probability density function for n_r. The distribution reflects user preference to consecutively access data with similar data properties.

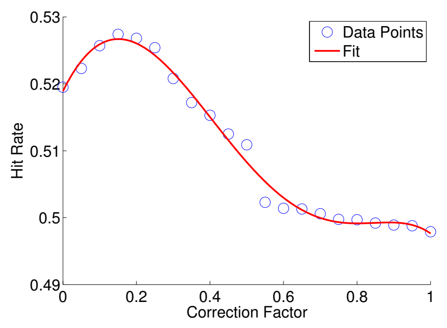

Fig. 3. Plot of average hit rate against varying values of w_c. There exists no other relative maximum for $0 \leq w_c \leq 1$.

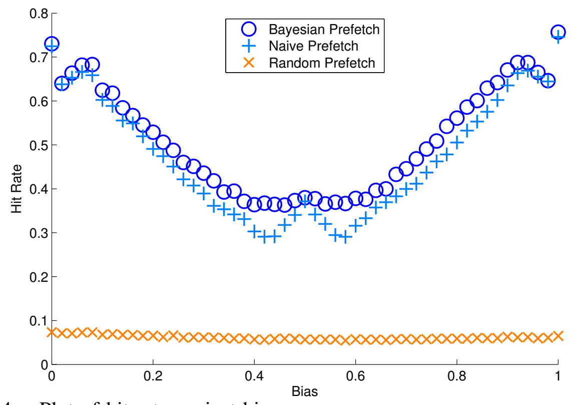

Fig. 4. Plot of hit rate against bias.

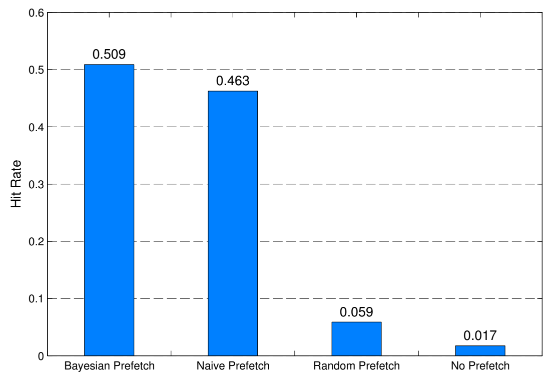

Fig. 5. Simulation-derived hit rates for bias$= 0.76$.

A total of 9 simulation cycles are run to achieve a 95% confidence interval. Each simulation cycle involved 144,000 accessed nodes distributed across 480 generated RSs. Total running time for the simulation was close to 45 hours, with 5 hours per cycle. The hit rate for each w_c value is then averaged over all 9 cycles, and plotted against w_c as shown in Figure 3. The graph is fitted with a polynomial for interpolation. The optimal weight correction value thus corresponds to the relative maximum, i.e., $w_c^o = 0.15$.

C. Simulation-based Performance Evaluation

We evaluate the performance of all three prefetch policies, viz. Naïve, Random and Bayesian, using our simulation framework. The performance of each of the prefetch policy is measured using the **hit rate**, which evaluates the effectiveness of the algorithm in correctly predicting the next data retrieval $\mathcal{P}_{|RS|+1}$.

The optimal value of the weight correction $w_c^o = 0.15$ is used for the Bayesian prefetch algorithm. In each simulation cycle, the value of the bias Q is varied from 0 to 1, in step size 0.02. A total of 9 simulation cycles were executed for each prefetch policy.

Figure 4 illustrates the average hit rate across each value of bias. Generally, the Random prefetch offers insignificant hit rates of approximately 0.06 across all values of the bias, thus highlighting the inefficiency of random prefetching with typical user behaviour. Due to the intrinsic random nature of the Random prefetch algorithm, it is agnostic to any bias in the generated request sequence. Such a random scheme is typically not used in actual implementations, but we include it as a baseline comparison. In contrast, both Bayesian prefetch and Naïve prefetch have higher hit rates, which is a key benefit of prefetching.

The Bayesian policy consistently outperforms both the Naïve and Random prefetch policies across all values of the bias. On average, the Bayesian prefetch policy achieves a higher hit rate of 0.04 as compared to the Naïve prefetch policy. The largest performance improvement is observed in the region $0.4 \leq$ bias $Q \leq 0.6$. As the bias approaches 0 or 1, the performance gap between the Bayesian prefetch and the Naïve prefetch decreases. This observation is expected as the Naïve approach prefetches based on the bias Q in the request sequence; hence, the more extreme the bias, the more accurate the predictions will be.

Figure 5 shows the hit rate performance of all three prefetch protocols when the bias $Q = 0.76$. Recall that this value of the bias is the statistical average of the logged data usage patterns of actual users. With this expected usage pattern, the Bayesian prefetch outperforms both the Naïve prefetch and Random prefetch baseline comparisons. With the higher hit rates achievable by the Bayesian prefetch, the delay experienced by users during data retrieval will decrease correspondingly, leading to higher user satisfaction.

V. Trace-Based Evaluation and Discussions

A. Trace-based Performance Evaluation

The final evaluation of the proposed Bayesian algorithm is accomplished through actual traces of data request usage. All

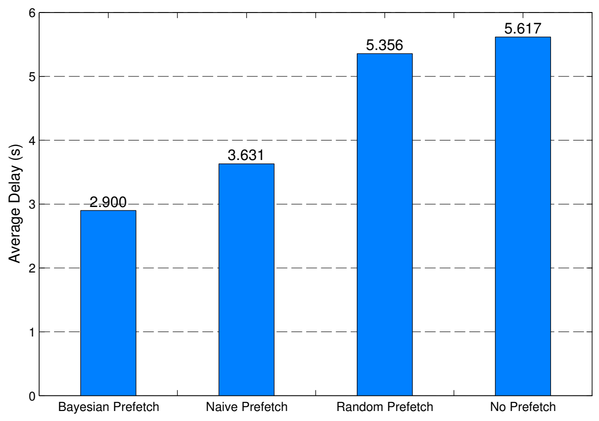

Fig. 6. Real-World delay time for each prefetch mode

the three prefetch algorithms (Naïve, Random and Bayesian) are incorporated into *Sensorem*, our mobile visualization platform for wireless sensor networks. Data request usages of eight network management personnel were logged through *Sensorem*, for a total of four maintenance runs spanning one day each. Two of the usage logs had Bayesian prefetch activated, two had Naive prefetch activated, two had Random prefetch activated, and two had prefetch deactivated. A total of 803 nodes were accessed during the runs.

Based on the results in Figure 6, it can be seen that Bayesian prefetch offers the greatest reduction in average delay incurred, amongst the three prefetch policies. The results show a 48.4% reduction in average delay using the Bayesian prefetch, in comparison with the 35.4% and 4.66% reductions yielded by the Naïve and Random policies. The improvements in performance brought about by the Bayesian algorithm can be attributed to its ability to adapt to user styles, and make intelligent inferences based on appropriate real-time feedback.

The significant delay reductions illustrate the effectiveness of prefetching and caching, in providing good user experience, despite the presence of backend bottlenecks. This is an invaluable performance and scalability trait that will enhance the applicability of *Sensorem* in future large-scale smart city applications.

B. Discussions

An important observation is that the Bayesian prefetch outperforms the Naïve prefetch much more significantly in the trace-based evaluations, as compared to the simulation-based evaluations. We attribute this to the absence of certain human behavioural traits in our characterization of user behaviour for Request Sequence generation algorithm. For instance, certain behavior - such as the tendency of human users to access certain sequences of nodes preferentially - cannot be captured in the RS generation algorithm.

Another issue of concern is that actual values of T_s and T_u fluctuate noticeably through standard application usage, with standard deviations of 0.96 and 5.14 respectively. Large fluctuations in T_s and T_u may affect the number of sensors prefetched and cached during each prefetch session, thereby cumulatively affecting hit rates. This effect is not accounted for in our optimization simulation. This issue may, in the future, be addressed by statistically determining the distribution of T_s and T_u values and incorporating the distributions into the simulation.

VI. Conclusion

This paper proposes a Bayesian prefetch algorithm to improve the scalability of data retrievals in large-scale smart city applications. The algorithm comprises a dynamic weight adaptation feature, that enables better prediction of the prefetched data, according to actual data usage patterns. The proposed Bayesian prefetch algorithm has been compared against the Naïve and Random prefetch policies. Both simulation and trace-based performance evaluations show that the Bayesian approach is the most effective of the three prefetch algorithms in reducing user-perceived delays during data retrieval.

As part of future work, the Bayesian prediction algorithm can be extended to accommodate multiple data property values for better prediction. In addition, alternative weight adaptation policies employing non-linear correction policies can be explored.

Acknowledgement

This project is completed as part of the Science Mentorship Programme in Singapore, in collaboration with the Singapore University of Technology and Design and the Institute for Infocomm Research, A*STAR.

References

[1] Department of Economic and Social Affairs, United Nations, “World urbanization prospects: The 2011 revision,” 2011.

[2] M. Batty, K. W. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali, “Smart Cities of the Future,” *The European Physical Journal Special Topics*, vol. 214, no. 1, 2012.

[3] J. Jin, J. Gubbi, T. Luo, and M. Palaniswami, “Network architecture and QoS issues in the internet of things for a smart city,” in *ISCIT*, 2012.

[4] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, “A study of integrated prefetching and caching strategies,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 23, no. 1, 1995.

[5] V. N. Padmanabhan and J. C. Mogul, “Using predictive prefetching to improve world wide web latency,” *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, 1996.

[6] J. Domenech, A. Pont, J. Sahuquillo, and J. Gil, “A comparative study of web prefetching techniques focusing on users perspective,” in *International Conference on Network and Parallel Computing*, 2006.

[7] A. Parate, M. Bhmer, D. Chu, D. Ganesan, and B. Marlin, “Practical prediction and prefetch for faster access to applications on mobile phones,” in *UbiComp*, 2013.

[8] C. Aggarwal, J. L. Wolf, and P. S. Yu, “Caching on the World Wide Web,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, 1999.

[9] J. M. Koh, M. Sak, H. Tan, H. Liang, F. Folianto, and T. Quek, “*Sensorem* - an efficient mobile platform for Wireless Sensor Network visualisation,” in *International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2015.

[10] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, “Informed prefetching and caching,” *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5, 1995.