Hybrid Dissemination Based Scalable and Adaptive Context Delivery for Ubiquitous Computing

Lenin Mehedy, Md. Kamrul Hasan, Young Koo Lee[1], Sungyoung Lee, Sang Man Han

Real Time & Multimedia Lab, Department of Computer Engineering,
Kyung Hee University, 449-701, Republic of Korea
{lenin, kamrul, sylee, i30000}@oslab.khu.ac.kr,
[1]yklee@khu.ac.kr

Abstract. Context delivery is an inevitable issue for ubiquitous computing. Context-aware middlewares perform all the functions of context sensing, inferring and delivery to context-aware applications. But one of the major issues for these middlewares is to devise a context delivery scheme that is scalable as well as efficient. Pure unicast or pure broadcast based dissemination can not provide scalability as well as less average latency. In this paper we present a scalable context delivery mechanism for context-aware middlewares based on hybrid data dissemination technique where the most requested data are broadcasted and the rest are delivered through unicast. Our scheme is adaptive in the sense that it dynamically differentiates hot (most requested) and cold (less requested) data according to request rate and waiting time. Inclusion of lease mechanism and bandwidth division further allows us to reduce network traffic and average latency. We validated our claim through extensive simulation.

1 Introduction

Context awareness is the ability to sense, interpret and respond to the situation of an entity (e.g. user, application) [2]. Often, the term "Context-aware Computing" is used synonymously to Mark Weiser's revolutionary concept of Ubiquitous Computing [1] as every future application will have context-awareness. Context-aware middleware performs all the functions of context sensing, inferring and then delivers to smart applications (see [4] for a survey). Thus every context-aware middlewares usually have the following three main phases of execution: a) Acquisition of raw sensor data, b) Context inference from the sensory data, c) Context delivery to applications. This paper focuses on the context delivery mechanism for such middlewares.

Our work is motivated by the unavoidable need of scalable context delivery in large smart environments (such as a corporate office, academic building, shopping complex etc.) where numerous context-aware applications (we interchangeably use client or receivers), running on mobile devices like PDA or stationary devices (desk-

[1] Dr. Young Koo Lee is the corresponding author

top PC), frequently request for various contexts to the middleware. If the context information of interest is same among different clients, traditional unicast (point to point or pull based) connection-oriented data services are uneconomical because it incurs a lot of unnecessary traffic from clients to server and vice versa. Even if the current technology allows us to have high network bandwidth and server capacity, most of it would be underutilized and wasted during non-peak periods. Broadcast (push based) is an efficient and scalable dissemination method in a connection-less mode to any number of clients with no significant performance degradation in terms of access latency[5]; but a major concern for the success of such system is broadcasting the right set of data. Because, broadcasting less important data may cause network overload. On the other hand, in on-demand broadcast (pull-push) method, the server aggregates the requests of clients and broadcast the data. But broadcasting the context with lowest request rate (cold item) may also increase network traffic. So hybrid approach combines the benefit of broadcasting hot context data (having higher request rate) and that of unicasting the cold context data (having lower request rate) [6]. Even with this suitable and scalable approach, we have the problem of differentiating hot and cold context data and formulate a suitable delivery mechanism for quick response time.

By the term "efficient delivery" we mean that the mechanism should cause less network traffic, provide quick response time and deal with dynamic nature of ubiquitous environment (i.e. clients appear and disappear in an unpredictable manner). Unavailability of such comprehensive solution motivates us to propose a context delivery mechanism for context aware middlewares in ubiquitous computing domain that is scalable as well as adaptive to request pattern. Our solution is an effort towards developing a robust and comprehensive context delivery mechanism for the middleware CAMUS (Context Aware Middleware for Ubiquitous System) [3], [17].

This paper is organized as follows: Section 2 describes related works. Section 3 explains the proposed method of context delivery. Section 4 presents the performance evaluation and Section 5 concludes with some future works.

2 Related Work

To the best of our knowledge, the researches in context delivery of the middlewares have not addressed so far the scalability issue where contexts are to be efficiently delivered to large number of mobile or static clients in ubiquitous environment. The most related research to ours is the Context Discovery Protocol (R-CDP) [13] that has been implemented and evaluated in "Reconfigurable Context Sensitive Middleware" (RCSM) [14]. The fundamental difference between R-CDP and our work is that R-CDP uses broadcast to request for a context and the middleware unicast the data to the requester, which is completely opposite to our mechanism as we use unicast for request and combination of broadcast and unicast for delivery. We use the technique of RxW algorithm [11] to prioritize the delivery where they use *Refresh Priority* that is based on the divergence of contexts and energy consumption of "Provider". We

also perform bandwidth division for hot and cold items for optimal average latency. The similarity with their work is that the motivation of their *Neighbor Validation Beacons (NVB)* is same as that of our Lease Renewal and we also have the way of specifying the *update threshold* for context update notification. However, they do not use context ontology for semantic interpretation. Moreover, R-CDP has not been tested for scalability [13], which we believe an important performance issue for large scale deployment of smart applications.

The idea of Hybrid data dissemination technique was first used in the Boston Community Information System [8] by combining broadcast and interactive communication to provide most updated information to an entire metropolitan area. This scheme is also adopted in [5], [6], [7], [8], [9], and [10] where the issue of mixing push and pull web documents together on a single broadcast channel was examined. But the document classification problem was introduced in [6] and later document classification along with bandwidth division was resolved in [7]. We employ these ideas of hybrid dissemination, scheduling, classification of data, bandwidth division etc. for scalable and efficient delivery of context for ubiquitous computing environment. Though our approach is very similar to [7], but we differ in calculating the popularity of an item not only on the total request but also on the longest waiting time of any outstanding request according to RxW as RxW does not suffer from starvation of request for cold item [11]. Moreover we also employ lease mechanism to reduce the periodic request (polling).

3 Proposed Scheme

Before introducing our context delivery scheme, let us assume that the clients request context information using context ontology for semantic interoperability (e.g. Contel [12] [17]). We use hybrid dissemination scheme [6] for the context delivery in ubiquitous environment. But this scheme introduces three inter-related data management problems at the middleware: First: the middleware must dynamically classify the requests between hot and cold context data and schedule the delivery according to priority or popularity (*Prioritization*). Second, the middleware should divide dynamically its bandwidth between unicast pull and multicast push for optimal use of bandwidth to ensure low *average latency* (*Bandwidth Division*). Third, as the hot context data are broadcast, some clients may receive the information passively in the sense that they do not send any request for that data and we call them "passive client". Therefore, the middleware lacks a lot of invaluable information about the data requirement and fails to appropriately estimate the hotness and coldness of data items (*Push Popularity Problem [7]*). The *average latency* for a data item on the push channel is half of the period of the broadcast cycle if we assume that the items are broadcast sequentially. However, the latency for pulled items are totally different because if an item i of size S_i is queued at the server for transmission, the corresponding queuing delay is either $O(S_i)$ or unbounded [7].

3.1 Message Format

All the clients request for context information according to the context ontology. Clients specify the type of context (e.g. Temperature) and as well as the entity of which this context is related (e.g. Room). "Lease Duration" and "Update Threshold" are also to be specified if a client needs a context for a certain amount of time to avoid polling. Thus the *Request* message contains the following information: Context Type (CT), Entity Type (ET), Entity Id (EID), Lease Duration (LD) and Update Threshold (UT) (see Table 1). If any client does not need periodical update notification, it should specify the "Lease Duration" field and "Update Threshold" field as zero. *Reply* message contains CT, ET, EID, value (V), Maximum Lease Duration (MLD), Minimum Update Threshold (MUT) and Report Probability (RP). RP is discussed in section 3.4 in detail.

Table 1. Message Format

Type	Content
Request	{CT, ET, EID, LD, UT}
Reply	{ CT, ET, EID, V, MLD, MUT, RP }

3.2 Prioritization

The middleware enqueues the requests according to different context groups and maintains the following information for each of the groups:

- Total Leased Request (TLR): Total number of leased requests for this specific context. This value is used in determining whether this data should be scheduled for broadcast (hot item) or unicast (cold item).
- Leased Request List (LRL): This list contains the ids (IP address) and lease durations of leased clients.
- Max Lease Duration (MLD): Maximum lease duration among the leased duration. This information is also sent along with the data to let the clients know how long this data will be delivered. If any client is receiving the data passively and wants to use longer than this time, it will renew the lease for longer duration.
- Total Pending Request (TPR): This field denotes the total number of requests that have been received but no delivery of the context has been done yet. This field is reset to zero after each delivery of the context and incremented after receiving of each new request for this context. The larger the value of this field, the higher the priority of delivery of this context should be.
- Pending Request List (PRL): This list is similar to LRL but contains the requesters' ids (IP address) and requested lease duration of the pending requests. After the delivery, the requests with lease duration greater than zero will be added to the LRL and TLR will be updated accordingly.
- First Arrival Time (FAT): This is the arrival time of the first request which is still pending. Longest waiting time (LWT) of any pending request for this context is the difference of current time and FAT. FAT is reset to zero after

each delivery and set to the arrival time of the first request as it is queued. The larger the value of this field, the higher the priority of this context should be for delivery.

- Last Delivery Time (LDT): The most recent time when the context was delivered. The difference between current time and LDT is the longest waiting time of the leased clients.
- Min Update Threshold (MUT): The minimum of update threshold values among the requests**.** If the context is changed by this amount, it is then scheduled for delivery.
- Candidate for Scheduling (CS): This is a binary value. If the amount of change exceeds the Min Update Threshold (MUT), the value of this field becomes one (true) and implies that this data should be delivered. This field becomes zero (false) with the next delivery of the context data.

To set the priorities of the requested context data, we use the total number of requests (R) and longest waiting time of the outstanding request (W) for that item and it is motivated by RxW algorithm [11]. In RxW algorithm, the item with higher R*W value has higher priority. Thus we prioritize a data either because it is very popular or because it has at least one long-outstanding request. We consider both Total Pending Request (TPR) and Total Leased Request (TLR) when CS is one (true) to be the total number of request (R), otherwise we only consider TPR to be the R value for the context item (see equation 1). This is because TLR comes into account as soon as the amount of change exceeds Min Update threshold (MUT) and CS becomes one (true). Similarly, as long as CS is zero, difference of current time (CT) and FAT (First Arrival Time) is the value of longest waiting time (W); but as soon as CS becomes one, the longest wait time (W) is the difference of CT and LDT (Last Delivery Time) as all the leased requests have been waiting since LDT. Hence we define RxW with the following equation:

$$RxW = \left(TPR + CS * TLR\right) * \left(\left(CT - FAT\right)\left(1 - CS\right) + CS\left(CT - LDT\right)\right) \quad (1)$$

We calculate RxW value of each group (data item) and sort them in descending order. We update the list each time a request comes and use the same data structure proposed in [11] for efficient maintenance of such list. The RxW value of i th group is denoted as popularity (or priority) p_i in the following sections.

3.3 Bandwidth Division

The motivation of bandwidth division comes from the fact that the *average latency* (L) of a data item is less when hot items are assigned to broadcast, cool items to unicast pull and the bandwidth is divided appropriately between the two channels. We used the bandwidth division algorithm similar to that is suggested for web server in [7]. But, we use the prioritization described in section 3.2 rather than using the prioritization based on request rate only as it is used in [7]. The bandwidth division algo-

rithm uses the sorted list of items with decreasing order of popularity, i.e. $p_i \geq p_{i+1}$ $(1 \leq i \leq n)$, where n is the current size of the list. It is intuitive that if item i is pushed, then $j \leq i$ should also be pushed. So, the algorithm tries to partition the list at index k such that the push set $\{1, 2, ..., k\}$ minimizes the latency L given a certain bandwidth B. The optimal value k^* is found by trying all possible values of L and finally the algorithm determines the pull bandwidth $\alpha \sum_{i=k+1}^{n} \lambda p_i S_i$, which leaves bandwidth $pushBW = B - \alpha \sum_{i=k+1}^{n} \lambda p_i S_i$ for the push channel and average latency for the pushed documents is then $\sum_{i=1}^{k} \frac{S_i}{2\,pushBW}$. Here λ and S_i denote request rate and size of the item respectively. The algorithm runs in $O(n)$ as it performs binary search over all possible values of L and maintains an internal array that stores the total size of each possible partition using binary tree techniques [7].

3.4 Push Popularity Problem

As the data is broadcast, some clients may not need to send any explicit request. This will misguide the middleware to identify most requested item and thus it introduces the "*push popularity problem*". We can not expect to solve this push popularity problem completely as it will require all the clients to send requests explicitly and hinder the benefit of broadcast. So, a portion of the clients that are passively accessing data should send requests even though the data is ensured to be delivered. The middleware sends a report probability (RP) with the data and client submits an explicit request for this data with probability RP. It is proved in [7] that RP should be set inversely proportional to the predicted access probability for that data and the equation to calculate RP is:

$$RP_i = \begin{cases} \frac{\beta}{\lambda p_i k}, \text{ if } \lambda p_i k > \beta \\ 0.2, \text{ otherwise} \end{cases} \quad (2)$$

Where β is the difference of Maximum acceptable TCP connection and request arrival rate, λ denotes aggregate request rate and p_i denotes the priority (or popularity) of group i based on total request and k denotes the current number of broadcast items. Here we notice that if $\lambda p_i k < \beta$, the probability will exceed one and hence we specified RP to be 0.2 as a default. It should also be noted that whenever the client sends a request, it sends a complete request with its desired update threshold (UT) and desired lease extension (LE). LE denotes the desired extension after the expiry of

current MLD. The same request format is also used for lease renewal when the lease expires.

4 Performance Evaluation

In order to establish the potential of our proposed context delivery mechanism, we have built a simulation model of the proposed system and evaluated using the simulation tool OMNET++ [16]. All the graphs presented here are generated using the PLOV tool of OMNET++.

4.1 Simulation Model

In our client-server model the server (our middleware) acts as a data server and delivers self identifying context data items of equal size either by broadcast or unicast upon explicit request. The clients request an item according to Zipf distribution [15] and the time of requests is exponentially distributed with mean M , where $\frac{1}{M}$ is the average request rate of each client. We present the analysis of average latency and network traffic of the proposed system in the following sub-sections. Table 2 presents all the simulation parameters. Here the pull over-provisioning factor α and the tolerance factor ε are used by the bandwidth division algorithm described in [7].

Table 2: Simulation Parameters

Parameter	Value
Total Client	3000
Total Item, N	50
Size of Each Item	200 bytes
Zipf parameter Θ	1.5
System Bandwidth	512,000 bits/sec
Exponential mean M	12
α	2
ε	0.005
Lease Duration of a client	10~ 100 ms (uniform)
Lease Renewal Probability	0.7
Update Threshold	0.5~2 unit (uniform)

4.2 Average Latency

Let $G(k)$ be the average latency (T_{avg}) if the k most popular items are broadcasted. The function $G(k)$ is a weighted average of the average latency of pushed items $T_{push} = \sum_{i=1}^{k} \frac{S_i}{2\,pushBW}$ and the average latency for the pulled

items $T_{pull} = \frac{1}{\mu - \left(\sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{k} \lambda_i \right)}$, where λ_i is the Poisson request rate for each item i [6], [7]. Our result is shown in Fig 1. Notice that the minimum of $G(k)$ is to the left of the intersection (at k=10) of the push and pull curves though theoretically it should be on the right side of the intersection [6]. The minimum of $G(k)$ occurs at a relatively small value of k and precedes the intersection due to two complementary reasons. First, the most popular items are chosen for push and are also those to which a Zipf distribution gives substantially more weight. So, if an item is delivered using broadcast, it will also have the largest impact on the globally average delays. As the numbers of the most popular items are small and are broadcasted first, the overall minimum delay occurs for small values of k. Second, pull delays are actually minimized at the points k' where the pull-curve flattens out. However k' precedes the intersection in our graph, and so the overall minimum occurs before that intersection.

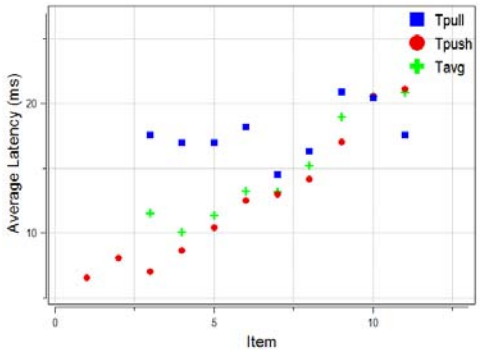

Fig 1: Relation of average latency of Push and Pull as the number of broadcast items changes according to our experiments. Here the intersection of T_{push} and T_{pull} occurs at $k = 10$ and before k=3, T_{pull} grows arbitrarily large.

4.3 Network Traffic

Fig 2 and Fig 3 presents our simulation result regarding network traffic. Here we can see in Fig 2 that in the beginning of time, the number of request is high. But as the server starts to deliver items, the number request decreases due to two reasons. First, the replies contain the maximum lease period and minimum threshold value for the context items and the clients do not need to send explicit request again until the lease expires. Second, as the most popular items are broadcast, the clients that also need the data do not send request but uses the data passively. But we can see some spikes in the request graph because of the lease renewal requests and the requests sent by the passive clients due to *Report Probability* as we have already discussed. Here we see

that incorporation of lease mechanism and threshold reduces overall network traffic from client as well as from server (Fig 3).

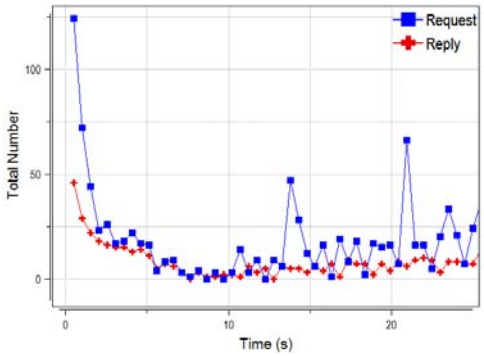

Fig 2: Number of request and reply with change of time using our approach. The number of reply denotes total number of broadcast and unicast items.

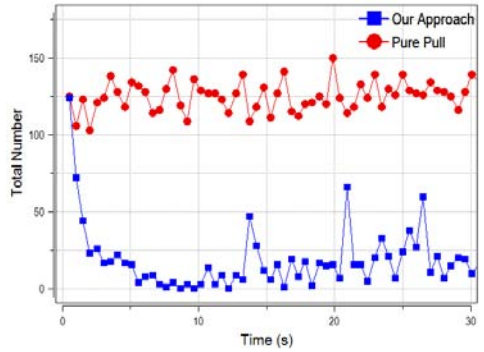

Fig 3: Number of requests in our approach and in pure pull approach. Simulation results show that our approach causes fewer requests as it avoids polling and uses lease mechanism.

5 Conclusion and Future Work

In this paper we present a scalable context delivery mechanism for context-aware middlewares based on hybrid data dissemination technique where the most requested data are broadcasted and the rest are delivered through unicast. Bandwidth division and lease mechanism are two notable properties of our approach that reduce average latency and network traffic respectively. Correlation of contexts, real time delivery, predictive broadcast, secure delivery etc. are some of the interesting approaches to extend this work.

Acknowledgement. This work was supported by IITA Professorship for Visiting Faculty Positions in Korea from Ministry of Information and Communications.

Reference

1. Weiser, M.: The Computer for the 21st Century. In: Scientific America, Sept. 1991, pp. 94-104; reprinted in IEEE Pervasive Computing, 2002, pp. 19-25
2. Dey, A.K, Abowd, G.D.: Towards a Better Understanding of Context and Context-Awareness. In Proc. of the 2000 Conference on Human Factors in Computing Systems, The Hague, The Netherlands, (April 2000)
3. Hung, N.Q., Shehzad, A., Kiani, S.L., Riaz, M., Lee, S.Y.: Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework. In: Proc. of Embedded and Ubiquitous Computing: EUC 2004, LNCS Volume 3207, Springer-Verlag (2004), pp. 672 – 681
4. Baldauf, M., Dustdar, S.: A Survey on Context-aware systems. Int. J. of Ad Hoc and Ubiquitous Computing, forthcoming
5. Acharya, S., Franklin, M., Zdonik, S.: Balancing push and pull data broadcast. In *ACM SIGMOD*, (May 1997)
6. Stanthatos, K., Roussopoulos, N., Baras, J. S.: Adaptive data broadcast in hybrid networks. In the 23rd Int. Conf. on VLDB, 30(2), (Sep. 1997)
7. Beaver, J., Morsillo, N., Pruhs, K., Chrysanthis, P. K.: Scalable Dissemination: What's Hot and What's Not. In the Seventh Int. Workshop on the Web and Databases (WebDB 2004), Paris, France, June 17-18 (2004)
8. Gifford, D.: Polychannel Systems for Mass Digital Communications. *CACM*, 33(2):141-151, (Feb. 1990)
9. Hall, A., Taubig, H.: Comparing push- and pull-based broadcasting or: Would "microsoft watches" profit from a transmitter? Lecture Notes in Computer Science, 2647 (Jan. 2003)
10. Triantafillou, P., Harpantidou, R., Paterakis, M.: High performance data broadcasting systems. Mobile Networks and Applications, 7 (2002) 279–290
11. Aksoy, D., Franklin, M.: RxW: A scheduling approach for large-scale on-demand data broadcast. IEEE/ACM Transactions On Networking, 7(6) (Dec. 1999) 846-860
12. Shehzad, A., Hung, N.Q., Pham, K.A., Lee, S.Y.: Formal Modeling in Context Aware Systems. In Proc. of Workshop on Modeling and Retrieval of Context, CEUR, ISSN 613-0073, Vol-114, Germany (2004)
13. Yau, S.S, Chandrasekar, D., Huang, D,: An Adaptive, Lightweight and Energy-Efficient Context Discovery Protocol for Ubiquitous Computing Environments. In the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)
14. Yau, S.S, Karim, F, Wang, Y, Wang, B., Gupta, S.: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. IEEE Pervasive Computing, 1(3), July-September (2002), pp.33-40.
15. Wentian Li, References on Zipf's Law. URL: http://www.nslij-genetics.org/wli/zipf/
16. OMNET++, URL: http://www.omnetpp.org/index.php
17. Shehzad, A., Hung N.Q., Anh, K.P.M. , Riaz, M., Liaquat, S., Lee, Y.K, Lee, S.Y: Middleware Infrastructure for Context-aware Ubiquitous Computing Systems. CAMUS Technical Report (TR-V3.2). February 2005. http://oslab.khu.ac.kr/camus