2011 Fourth IEEE International Conference on Utility and Cloud Computing

Towards Context Caches in the Clouds

Saad Liaquat Kiani*, Ashiq Anjum†, Kamran Munir*, Ricahrd McClatchey* and Nick Antonopoulos†
**Faculty of Engineering and Technology, University of the West of England, Bristol, UK*
†*School of Computing and Mathematics, University of Derby, Derby, UK*

***Abstract*—Context information is traditionally collected from distributed digital artifacts and services and made available to similarly distributed, and often mobile, context consuming applications via context brokers or servers. Contextual data has a strong temporal element i.e. it remains valid for a period of time, and hence is an ideal candidate for caching strategies that aim to exploit such locality of reference. However, different types of contextual information have varying temporal validity durations and a varied spectrum of access frequencies as well. An ideal of the caching mechanism should utilize dynamic strategies based on the type of context data, quality of service heuristics and access patterns and frequencies of context consuming applications. This paper presents an investigation of the utility of various context-caching strategies and proposes a bipartite caching mechanism in a cloud-based context broker that facilitates context provisioning between context providing services and consuming applications. The results demonstrate the relative benefits of different caching strategies under varying context usage scenarios and the utility of the bipartite context caching mechanism in a context provisioning system.**

***Keywords*-cloud computing; context-aware clouds; context provisioning; intelligent caching; context-awareness**

I. Introduction

Context-aware systems facilitate the acquisition, representation, aggregation and distribution of this contextual information in ubiquitous environments. Established context-aware systems predominantly utilize a broker or a context server to facilitate context provisioning from providers of context information to context consumers. Due to the distributed nature of sensors and services that provide raw data for context creation, and that of applications/services that utilize such data, the provisioning of contextual information is a non-trivial task. Existing context-aware systems are mostly focused on small geographic and conceptual domains and the context provisioning function of these systems has not attracted in-depth attention. For instance, the temporal properties of contextual data are not utilized by existing context-aware systems to improve context provisioning performance through caching, grid and cloud based platforms. One of the key challenges in context-aware systems is the provisioning of contextual information about *anything, anytime and anywhere* [1]. Meeting this challenge requires an infrastructure that can reliably collect, aggregate and disseminate contextual information related to a very large user base over a large scale. Cloud computing is ideally placed to provide infrastructural support for meeting this challenge through its key characteristics of reliability, scalability, performance and cost effectiveness. However, context-aware systems have not yet taken advantage of this recent progress in the computing arena.

In addition to the intrinsic benefits of Cloud computing, contextual information itself has certain features that can aid in improving the performance of systems that deliver context information from context producing components to context consuming components. Context information remains temporally valid for a certain duration, which depends on the type of context data. This property of the context data can be exploited by employing context caches in context provisioning systems to improve the overall system performance as done routinely in distributed systems. Our motivation towards investigating this area builds on the observation that contextual data is central to the functional relevance of any context-aware system. With a significantly large number of users, devices, data sources and services involved in the end-to-end cycle of acquisition, reasoning, delivery and consumption of context, inadequate infrastructure support in terms of storage, processing, and provisioning of contextual information can be the biggest hurdle in adoption of context-aware systems over a large scale. Caching is a well established performance improvement mechanism in distributed systems, and if employed in Cloud-based context provisioning systems, can augment its infrastructure strengths and further improve the context provisioning function.

Context information is usually modeled using name-value pairs, software objects and structured or semi-structured records. Irrespective of the representation format, the context information has an ever-present temporal property i.e. the information remains valid for a certain period of time. For example, an instantiation of the *location* context of a user remains valid as long as the user remains in that location, the *weather* context of a user remains valid as long as the user remains within the geographic span whose weather information is quantified in a context instance, the *Wi-Fi* context of a device remains valid as long as the device is connected to a certain Wi-Fi hotspot. This temporal validity can be exploited in intermediate components of context-aware systems for improving the context-provisioning performance e.g. caching contextual data at a context broker can allow for the exploitation of the *locality of reference* in order to reduce contextual query satisfaction time and reduction in the overall context related traffic in the system.

978-0-7695-4592-9/11 $26.00 © 2011 IEEE
DOI 10.1109/UCC.2011.67

While caching is an established mechanism for performance improvement in distributed systems, the pertinent issues have not been analyzed extensively in the domain of context-aware systems. Firstly, different types of contextual data have varying validity durations i.e. a certain scope of context information (location, activity, Wi-Fi, weather, etc.) may remain valid for a few seconds while another scope may remain valid for days e.g. user-profile, device settings and shopping preferences. Secondly, the access rate and patterns of context consuming applications (distribution of scopes in context queries) may vary according to the time of day, type of context consuming application and user activity. Since caches are practically limited in size, cache replacement policies have to be employed and the variances in scope distributions in the queries, rate of the queries and validity periods of context scopes greatly influence the effectiveness of different replacement policies. The comparative effectiveness of different cache replacement policies needs to be analyzed and empirically evaluated.

Mere analysis of the caching strategies for contextual data provisioning is insufficient in the absence of a platform where their benefits can be fully utilized. The evolving technological landscape, characterized by increasing technological capabilities of smart devices and their adoption by everyday users, the greater availability of digital information services and the emergence of smart environments with embedded digital artifacts point towards an emerging digital ecosystem where a significantly large number of users in inter-connected smart environments will be utilizing context-based services through different computational interfaces. The success of context-aware systems will depend on accommodating these emerging scenarios and meeting their wide-spectrum requirements will greatly influence their adoption. Specifically, these requirements include device and location independence during utilization of contextual services, reliability of the system infrastructure, scalability in terms of load, administration and geographic scale, and the performance of the overall system in terms of query-response times and quality of service. A cloud based context provisioning system will 1) allow access to context information through standardised and interoperable interfaces, which will facilitate device and location independence, and 2) provide reliability and scalability through *elastic* and redundant resources. However, simply enabling Cloud based provisioning will not utilize the temporal validity characteristic of the context data, which can exploit the *principle of locality* to improve query-response times and therefore positively influence the quality of service of the context-aware system as a whole. Keeping these expectations in view, this paper relates the delivery of the caching functionality through a Cloud based system, but focuses primarily on establishing the suitability and relative effectiveness of different caching strategies for different types of contextual data. Once such effectiveness is established through experimental analysis, we aim to implement the strategies in a prototype Cloud based context provisioning system.

We discuss related work in Section II and describe the functional architecture of our Context Provisioning System in Section III. The experimental evaluation of the caching functionality, and that of cache replacement policies in context caches, is presented in Section IV. Based on the results of the experimental analysis, we propose a novel caching strategy for utilization in context provisioning systems and discuss its dynamic re-configuration based operation as well. The paper is concluded in Section V with a description of relevant points that direct the future of this preliminary work.

II. Related Work

A number of server/broker-based context provisioning systems have been developed, e.g. CoBrA [2], JCAF [3] and MobiLife [4] but caching contextual information has not been targeted in these systems explicitly. The MobiLife architecture specifies context caching at the context provider component but this approach creates distributed context caches at each context provider, potentially saving computational load at the providers but not reducing the communication cost. The query from the context consumer has to traverse the complete round trip from the context provider via the context broker. This mechanism can be improved by building a collective cache based on the smaller caches at context provider level. Buchholz et al. [5] discuss the importance of caching context in improving its quality. Ebling et al. [6] also highlight caching of context data as an important issue in designing context-aware services. Caching context requires that the validity of a context data instance can be specified. This can be achieved by the inclusion of temporal information in the context representation format. MobiLife is one of the few context-provisioning systems that specify a caching component at the architecture level. However, its context representation format contains no metadata that specifies its temporal properties. A similar system is the Context Based Service Composition (CB-SeC) [7] that employs a Cache Engine for providing context based service composition. However, the CB-SeC system does not store context information but the whole context service in the cache. A Caching Service is demonstrated in the SCaLaDE middleware architecture [8] for use with Internet data applications. The focus of this Caching Service is on providing disconnected operation to mobile devices by keeping a mobile device cache consistent with a central cache in the network. However, no performance metrics are reported regarding the gains achieved by the use of this cache. Despite the established significance and usability of caching components in distributed systems, context aware systems have not, as yet, demonstrated their use. Some researchers have highlighted the importance of caching context information but no study has reported any results on the empirical gains of employing a context cache in a context provisioning system and this deficiency has served as the main motivation for our continuing study of this domain. The discussion presented in this paper builds on our earlier work that demonstrated one of the first empirical studies on caching contextual data in context provisioning systems [9].

III. The Context Provisioning Architecture

Our system is based on the producer (provider)-consumer model in which context related services take the roles of context providers or context consumers. A Context Consumer (CxC) is a component that uses context data and retrieves context by sending subscriptions to the Context Broker (CxB). The Context Provider (CxP) component provides contextual information. A CxP gathers data from a collection of sensors, network/cloud services or other relevant sources and is usually specialized in a particular context domain (e.g. location). A Context Broker (CxB) is the main coordinating component of the architecture that facilitates context flow among all attached components, which it achieves by allowing CxCs to subscribe to or query context information and CxPs to deliver notifications or responses. Further details of this architecture are described in [10].

CxCs and CxPs register with a CxB by specifying its communication end point and the type of context they provide or require. This in turn enables a brokering function in which the context broker can look up a particular CxP that a CxC may be interested in (e.g. based on the type of context being requested). The broker can cache recently produced context, in order to exploit the principle of locality of reference. A distinguishing feature of this architecture is the federation of multiple context brokers to form an overlay network of brokers (Fig. 1), which improves the scalability of the overall system and provides location transparency to the local clients (CxCs and CxPs) of each broker. This federation is achieved with a coordination model that is based on routing of context subscriptions and notifications across distributed brokers, discovery and lookup functions and is described in detail in [11]. This concept of context broker federation can be directly related to Cloud federation in which two or more geographically distinct or administratively independent clouds cooperate in resource sharing and related functional operations, hence setting the conceptual foundation for federation of context-aware Clouds that exchange cross-domain context-information for serving their mobile/roaming users.

Context information is represented using an XML based schema entitled ContextML. The defining principle in ContextML is that context data relates to an *entity* (a username, a SIP or an email address etc.) and is of a certain *scope*. Furthermore, a temporal validity is associated with ContextML encoded context data through the *timestamp* and *expiry* tags, which specify the time duration during which a specific context instance is considered valid. This feature of ContextML forms the basis of utilizing the caching function

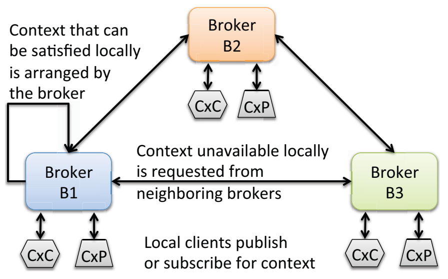
Context that can be satisfied locally is arranged by the broker
Broker B2
CxC
CxP
Context unavailable locally is requested from neighboring brokers
Broker B1
Broker B3
CxC
CxP
Local clients publish or subscribe for context
CxC
CxP

Figure 1. Simplified view of the federated broker based interaction

in the context brokers of the architecture. The model of the contextual data-related elements and a discussion about various dimensions of ContextML is presented in an earlier work [12]. Work is under progress to deploy the existing Context Provisioning Architecture on the Cloud infrastructure in order to exploit the scalability, reliability, performance and interoperability related benefits offered by the Cloud based systems. Figure 2 shows a conceptual diagram of how the system components may operate in a federation of context brokers in the Cloud infrastructure. Each context broker may operate under a different administrative authority but the federation between these brokers (and semi-private Clouds) can allow the context CxCs to utilize these brokers for acquiring contextual information. The federation features are beyond the scope of this paper and we will limit our focus to the specific feature of context caching in a single broker setup.

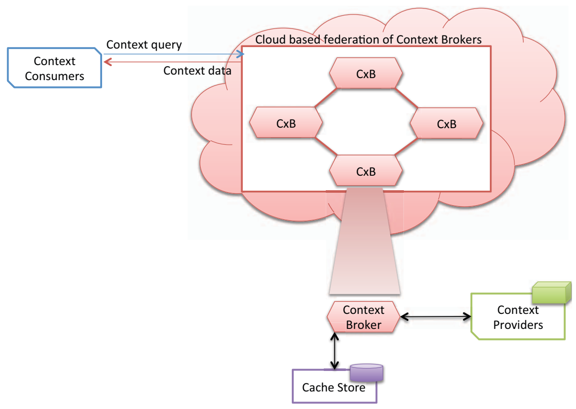

Figure 2. Architectural components of the Context Provisioning Architecture in the Cloud infrastructure

IV. Context Cache

CxCs request context about a particular entity and scope by forwarding a ContextML encoded query to the context broker. The broker forwards the query to an appropriate CxP that can satisfy it. When the query-satisfying context information is available, the CxP sends the context response to the broker. In the absence of a caching facility, the broker simply forwards the query to the querying consumer. The Context Provisioning Architecture utilizes a caching component that caches recently received contextual data in response to context queries, in addition to forwarding the response to the querying consumer. The context data remains in the cache for the validity period unless it is replaced by more recent context of the same scope or the entity has to be removed to free the cache due to cache size limits. In addition to the development and real-world deployment of the Context Provisioning Architecture system, a simulation model has been developed to evaluate the system under various conditions. The simulator is based on OMNET++, a Discrete Event Simulator toolkit, and models the actual system components, the representation scheme and the communication model as well. The results of the experiments carried out with this simulated setup will aid in establishing the suitability and relative effectiveness of caching strategies for context provisioning. These caching strategies can then be readily implemented in a Cloud based context provisioning system to augment the reliability, scalability and device/location independence benefits that are provide by the Cloud setup.

A. Experimental Evaluation

The simulation model consists of a context broker module, CxPs and CxCs connected by communication channels. The simulator comprises the core functionalities of context caching, context querying service, CxP registration and lookup service. Furthermore, the ContextML schema is also fully modeled. CxP modules provide context on invocation by the CxB and provide context about one particular scope only. The simulation model comprises various input parameters that can be set individually for each simulation run allowing several scenarios to be evaluated and compared against each other. The parameters for each scope contain numerical scope ID (integer) and its validity duration (seconds). The parameters for each Context Provider comprise a CxP ID (integer), ID of the context scope that it provides (integer) and the average time taken to process a query and respond to it (ms). The context broker module parameters include the lookup time for finding CxPs for satisfying queries (ms), cache access time [ms], caching enabled (Boolean), maximum cache size (integer i.e. the number of items in the cache), and the cache strategy i.e. the cache replacement policy used (integer).

$$scopeID = \left\lceil maxScopeID \cdot \left(\frac{\xi}{randUniform\,(0,1]} \right)^{-\sigma} \right\rceil \qquad (1)$$

Within the scope of this evaluation, there are three main caching strategies that we will evaluate, including *remove oldest first* (OF*), remove least used first (LU), and remove soonest expiring first (SE)*, in addition to the non-practical strategy of having an infinite cache size thus requiring no replacement. We have already established the usefulness of caching contextual data in principle in our earlier work [9], but did not analyze the effect of variance in the scope validity durations in detail. As the results will demonstrate, different access patterns from users can have a significant influence on the performance of the cache. With the help of this simulation model, we intend to establish suitable strategies for varying access patterns and devise a caching strategy that can accommodate a combination of these access patterns. The CxCs are configured to request context a constant rate [/s]. The context scope specified by the CxCs in the queries is determined using a Pareto distribution with a selectable shape α and scale ξ (1). The discretized Pareto distribution has been selected because it allows us to model scope distribution in context queries with tuneable parameters. Twelve different scopes are used in this experiment and the scope distribution in context queries is controlled by changing the Pareto shape parameter α while the scale parameter ξ is kept constant at 1. In each simulation run 5000 context requests distributed across 10 entities are instantiated. After all the responses have been received by CxCs, the simulation is terminated. Scopes and CxPs are initialized using the values from Table I. The CxB cache access time and CxP lookup time are assumed to be 10ms. In each simulation run, a caching strategy is selected and the Pareto distribution for selecting the requested scopes (α) is varied to select a certain percentage of *short validity (SV)* and *long validity (LV)* category scopes. The simulation is repeated for each caching strategy and the query satisfaction time, the time elapsed between issuance of a query from a consumer and receipt of a response to that query, being recorded. Hence, the performance of the selected cache strategies is investigated with varying scope distribution in the context requests.

Table I
SIMULATION PARAMETERS

CxP:ScopeID	Processing time[ms]	Validity[s]	Category
CxP:1	70	60	Short
CxP:2	70	60	Short
CxP:3	80	80	Short
CxP:4	80	80	Short
CxP:5	90	180	Short
CxP:6	90	240	Short
CxP:7	70	360	Long
CxP:8	70	400	Long
CxP:9	80	600	Long
CxP:10	80	900	Long
CxP:11	90	1200	Long
CxP:12	90	1200	Long

B. Results

The mean query satisfaction times of 5000 context queries with different caching strategy are plotted in Fig. 3. We analyze the mean query satisfaction time of these caching strategies in the cases where scope distribution varies from being fully focused on short validity (SV) scopes to long validity (LV) scopes in increments of 25% i.e. the distributions range from (1.0 SV/0.0 LV), (0.75 SV/0.25 LV), (0.5 SV/0.5 LV), (0.25 SV/0.75 LV) and (0.0 SV/1.0 LV). The reference cases of having an unlimited and no cache show the maximum performance improvement possible with our setup. The mean query satisfaction time across different combinations of SV/LV scope distributions improves from 487ms to 292.8ms, with a cache-hit ratio of approximately 46%. However, an unlimited cache size is impractical, hence we focus our attention to various cache replacement policies that are evaluated with a fixed cache size of 500 items maximum i.e. 1/10th of the total number of context items that will be generated during an experimental iteration.

The caching sub-component in the broker keeps track of the number of times an item in the cache has found use i.e. cache-hits that have occurred. It also records the time of arrival of a context-item in the cache and time left in the expiry of an items validity. When space is needed in a full cache for a newer item, the LU cache replacement policy removes an existing item from the cache that has been accessed the least number of times. The chart in Fig. 3 shows that LU results in mean query satisfaction time of 358.8ms (with ˜33.5% cache-hit ratio) and provides a fairly even performance for both the short validity scope and long validity scope focused context queries. The *OF* cache replacement policy provides an improvement over LU with a mean query-satisfaction time of 341.8ms. However, it is evident by considering the results in Fig. 3 that *OF* delivers a better query satisfaction time when the scope distribution in the contextual queries is biased towards SV scopes. This can be explained by the fact that under a querying pattern where most of the queries contain requests for SV scopes, the SV context data items will dominate the cache store. But since these data items have shorter validity durations, by the time they are removed due to the *OF* policy they would be closer to the expiry instant and hence been offered a greater chance of generating a cache-hit by spending most of their validity period in the cache. In the reciprocal case of high concentration LV scopes in the context queries, *OF* policy results in the longer validity data items from the cache that are not often closer to their expiry instant and hence have not been offered a fuller chance to result in a cache-hit.

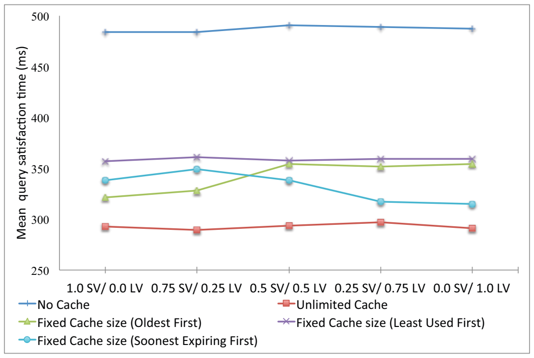

Figure 3. Mean query satisfaction times, different caching strategies and scope distribution scenarios

The *SE* cache replacement policy removes an item from the cache that is the closest to its validity expiration. This policy delivers an improved mean query satisfaction time of 331.4ms (with ˜36.25% cache-hit ratio) across all scope distributions but a closer inspection reveals that *SE* performs better for LV scoped queries than SV scoped queries. This policy is biased towards replacing an SV scoped item from the cache store because the validity expiry time for such items is more than likely to be closer than LV items. Moreover, an LV scoped removal candidate item would have spent most of its validity duration in the cache and thus given a good chance to result in a cache-hit.

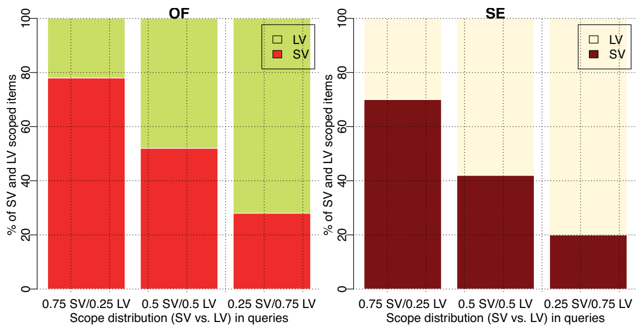

Figure 4. *OF/SE* replacement policies and cache-hit rate of SV vs. LV in different scope distribution scenarios

The *OF* and *SE* policies can be further examined by comparing the validity categories of data items that resulted in a cache-hit. Figure 4 illustrates the relative percentage of SV and LV scoped items in the cache-hit resulting items. Under the *OF* policy, SV scoped data items occupy a share of the context-hit space that is greater than their percentage in the context queries i.e. under the *OF* policy it takes a 78% share in the case of 0.75 SV/0.25 LV, 52% in case of 0.5 SV/0.5 LV and a 28% share in the case of 0.25 SV/0.75 LV. Contrastingly, under the *SE* policy, LV scoped data items occupy a greater share of the context-hit space i.e. under the *SE* policy it takes a 30% share in the case of 0.75 SV/0.25 LV, 58% in case of 0.5 SV/0.5 LV and a 80% share in the case of 0.25 SV/0.75 LV. These trends demonstrate the suitability of *OF* and *SE* policies for SV and LV scoped context data respectively. We have used these observations to devise a novel caching mechanism for contextual data that is suitable for both short and long validity scoped context data, which is discussed in the following section.

C. The Bipartite Context Cache

Taking into consideration the suitability of different cache replacement policies for SV and LV scope categories, we split the physical cache into two parts, one catering for the SV scoped context data items and the other for LV scoped items. The caching strategy is then configured to utilize *OF* replacement policy for SV scoped data and *SE* policy for LV scoped data items. The performance of the bipartite context cache is evaluated under the same experimental conditions discussed earlier and the results are plotted in Fig. 5. The bipartite cache provides a marginally improved overall performance over the *OF* and *SE*, with a mean query satisfaction time of 326ms (˜38.1% cache-hit ratio). This use of two different cache replacement policies suited to the scope validity durations of the data items results in improved performance and provides a fairly constant mean query satisfaction time across all scope distribution patterns. We have further evaluated the bipartite caching mechanism with a dynamic scaling of the size of the two partitions that is based on the distribution of scopes in the incoming context queries. Dynamically increasing or decreasing the size of a partition based on the ratio of a particular scope validity category in the incoming queries tunes the cache to accommodate the pattern of queries that exist in a particular situation. The query satisfaction times improve marginally under bipartite caching with dynamic partitioning from the case of equally sized bipartite cache. The mean query satisfaction time in our experiments is 318.4ms (˜39.4% cache hit-ratio) and the results display a consistent pattern across all scope validity scenarios (Fig. 5).

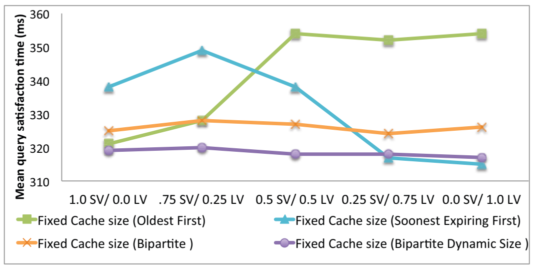

Figure 5. Inclusion of Bipartite with Dynamic Size cache strategy in the earlier comparison

V. Conclusions and Future Work

We have evaluated the relative performance of different caching strategies in a distributed context provisioning system. Our analysis has revealed that different caching strategies display contrasting behavior under different scope distribution scenarios, with *OF* policy performing better for short scoped context and *SE* performing better for long scoped context. Based on this observation, we have devised a novel bipartite caching strategy for context provisioning that allows utilization of the *OF* and *SE* policies for SV and LV scoped context data during context provisioning. The bipartite cache is further improved by allowing dynamic resizing of the cache partitions based on the scope distribution scenario of the context queries. The novel caching strategy can assist in designing a cloud based context provisioning system that *effectively* utilizes the temporal validity characteristic of the context data, exploit the principle of locality to improve query-response times and therefore positively influence the quality of service of the context-aware system as a whole.

The study described in this paper has been carried out using an OMENT++ based simulation model that mirrors a real-world deployment of the Context Provisioning System prototype. The experiments have been carried out under controlled conditions with a limited number of context scopes. Our primary future work is the integration and analysis of the bipartite caching mechanism in a cloud based deployment of the Context Provisioning System and carry out a large-scale analysis of the viability of the caching mechanism under real-world conditions.

References

[1] M. Weiser, “The computer for the twenty-first century,” *Scientific American*, vol. 265, no. 3, pp. 94–104, Sept 1991.

[2] H. Chen, T. Finin, and A. Joshi, “An intelligent broker for context-aware systems,” *Adjunct Proceedings of Ubicomp 2003*, pp. 183–184, October 2003, (poster paper).

[3] J. E. Bardram, “The java context awareness framework (JCAF) - a service infrastructure and programming framework for context-aware applications,” in *Pervasive Computing*, ser. LNCS. Springer, 2005, vol. 3468, pp. 98–115.

[4] P. Floreen, M. Przybilski, P. Nurmi, J. Koolwaaij, A. Tarlano, M. Wagner, M. Luther, F. Bataille, M. Boussard, B. Mrohs *et al.*, “Towards a context management framework for MobiLife,” *14th IST Mobile & Wireless Summit*, 2005.

[5] T. Buchholz, A. Küpper, and M. Schiffers, “Quality of context: What it is and why we need it,” in *Workshop of the HP OpenView University Association*, 2003.

[6] M. Ebling, G. D. H. Hunt, and H. Lei, “Issues for context services for pervasive computing,” in *Workshop on Middleware for Mobile Computing, Heidelberg*, 2001.

[7] S. K. Mostéfaoui, A. Tafat-Bouzid, B. Hirsbrunner *et al.*, “Using context information for service discovery and composition,” in *5th Intl. Conf. on Information Integration and Web-based Applications and Services*, vol. 3, pp. 15–17.

[8] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli, “A mobile computing middleware for location and context-aware internet data services,” *ACM Transactions on Internet Technology (TOIT)*, vol. 6, no. 4, p. 380, 2006.

[9] S. L. Kiani, M. Knappmeyer, E. Reetz, and N. Baker, “Effect of caching in a broker based context provisioning system,” in *Proceedings of The 5th European Conf. on Smart Sensing and Context*, vol. 6446. LNCS, Nov 2010, pp. 108–121.

[10] M. Knappmeyer, R. Tönjes, and N. Baker, “Modular and extendible context provisioning for evolving mobile applications and services,” in *18th ICT Mobile Summit*, 2009.

[11] S. L. Kiani, M. Knappmeyer, N. Baker, and B. Moltchanov, “A federated broker architecture for large scale context dissemination,” in *2nd Int’l Symp. on Advanced Topics on Scalable Computing*, Bradford, UK, June 2010.

[12] M. Knappmeyer, S. L. Kiani, C. Frá, B. Moltchanov, and N. Baker, “A light-weight context representation and context management schema,” in *Proceedings of IEEE International Symposium on Wireless Pervasive Computing*, May 2010.