2011 Third International Conference on Intelligent Networking and Collaborative Systems

Context Parsing, Processing and Distribution in Clouds

Saad Liaquat Kiani*, Ashiq Anjum†, Nik Bessis†, Richard Hill† and Michael Knappmeyer‡,*
*Faculty of Engineering and Technology, University of the West of England, Bristol, UK
†School of Computing and Mathematics, University of Derby, Derby, UK
‡Faculty of Engineering and Computer Science, University of Applied Sciences Osnabrück, Osnabrück, Germany

***Abstract*—Context is information that describes the situations in which computing, social and physical interactions take place. The complexity and scope of context information available for utilization by context-consuming applications, such as those executing on smart mobile devices, sensing and tracking platforms, etc. is growing with the increased integration of digital artifacts in smart environments. Similarly, the uptake of Cloud computing has significantly influenced the traditional information processing and infrastructure provision models by offering an agile, scalable and cost effective computing paradigm. This is leading to the adoption of Cloud-based solutions for the pervasive and ubiquitous environments; however, there are significant challenges that need to be overcome before its exploitation by real world applications and users. The context information consumed and produced by the applications and devices needs to be represented, disseminated, processed and consumed by numerous components in a context-aware Cloud system. Significant amount of context consumption, production and processing takes place on devices and there is limited or no support for collaborative modeling, persistence and processing between the device-Cloud ecosystems. In this paper we propose an environment for context processing in a Cloud-based distributed infrastructure that offloads complex context processing from the applications and devices. An experimental analysis of complexity based context-processing categories has been carried out to establish the processing-load boundary. The results demonstrate that the proposed collaborative device-Cloud infrastructure provides significant performance and energy conservation benefits for mobile devices and applications.**

***Keywords*-context processing; distributed infrastructure; Cloud computing;**

I. Introduction

Context-aware systems have distinct functional elements that perform the role of data acquisition, context synthesis, context storage, context dissemination and coordination. We label the functional roles of context dissemination and coordination as *context provisioning*. Under this functional task, the produced contextual information is communicated between context consuming applications and context producing services. The context *provisioning* function deals with making context information acquired by *context providers* available to *context consumers*. Context consuming and producing applications/services are usually assisted via *context servers or brokers* within the context provisioning systems to coordinate the flow of context queries and subscriptions and responses or notifications. A context provisioning system may also provide other services such as caching, reasoning and storage of contextual information to increase the overall functional usability of the system. Context-consuming applications usually aggregate simple context information (location, time, etc.) about entities (users, devices, etc.) into complex contextual information e.g. user activity. This aggregation may involve varying levels of reasoning and parsing of contextual data. While desktop based context-consuming applications have adequate processing ability available to for such purposes, the parsing, processing and reasoning about contextual information presents a processing burden to context-consuming applications deployed on smart mobile devices. Context servers or providers usually carry out the reasoning related processing, however the parsing, querying and processing of is still carried out by the context consumers. In this paper, we explore the possibility of delegating such tasks from smart device based context consumers to a Cloud-based service and evaluate the benefits of the approach in terms of processing time improvement and reduction in processing cost on such devices.

Our analysis of the focal issues is carried out within the scope of the Context Provisioning Architecture [1], which is a federated broker based system for provisioning of contextual information from Cloud-based context providing services to mobile and desktop based context consuming applications. The collection and dissemination of contextual information from Cloud-based services (context brokers and providers) contribute to the notion of context-aware Clouds. However, the consumption of context-information disseminated from the context-aware Clouds is expensive in terms of processing and communication costs. Modeling and representation of real world concepts into machine understandable format, for reasoning and consumption by applications, is a non-trivial task. Predominantly, semi-structured modeling approaches have been used for this purpose e.g. XML based schemas and Ontologies. Other, simpler approaches such as name-value pairs are easier to implement, but do not offer acceptable levels of expressiveness and the flexibility required for modeling real world concepts. The complexity involved in processing semi-structured data e.g. inference of useful information from an XML based document, is both time and space dependent. For example, the complexity of XPath query processing increases with respect to both the size of the query and the XML document containing

978-0-7695-4579-0/11 $26.00 © 2011 IEEE
DOI 10.1109/INCoS.2011.70

the information. Gottlob et al. [2] report that common implementations of XPath engines require exponential time with respect to the size of the queries in the worst case and that even the simplest queries take quadratic time in the size of the data. This complexity is significant in the domain of context provisioning systems in which context consuming and producing applications execute on smart mobile devices. Although the technological capabilities of such devices have increased rapidly in the last decade, they are still and likely to remain weaker as compared to desktop computing devices. Furthermore, another limitation is the limited energy available to battery-powered smart mobile devices and complex processing imposes an extra burden on the finite energy resources of the devices.

A natural solution to the issue of parsing, processing and distribution of context, is to offload such processing onto resource-capable computing systems such as those hosted in a Cloud infrastructure. But the additional communication steps incur further costs in terms of time taken to process information and energy used in communicating with Cloud hosted processing services. The querying complexity and document size boundary at which the offloading of query processing starts being beneficial in terms of time and energy costs needs to be investigated. Furthermore, offloading simple queries on small sized documents may not provide time/energy conservation benefits as compared to complex queries on large size documents. The relative benefits may also vary due to different processing capabilities of heterogeneous devices and platforms. This article is targeted towards addressing these issues and questions in our federated broker based Context Provisioning Architecture that utilizes an XML schema based model for contextual information. Specifically, our evaluation is focused on the processing of contextual information represented in ContextML [3], which is an XML based schema used for representation of control messages and context information in the Context Provisioning Architecture. The answers to the issues discussed above will provide an assessment of the degree of query complexity and document size to be *efficiently* managed by modern smart devices during context consumption and processing. Furthermore, our approach presents a mechanism for context provisioning within the Cloud infrastructure itself, supporting context-aware applications and services.

Before describing our context provisioning architecture and the experimental framework that is used for studying and evaluating the issues discussed above, we describe the background and work related to the problem domain being addressed in this paper in Section II. Description of the salient features of the federated broker [1], [4] based context provisioning system, including ContextML and Cloud based Context Processing Service is provided in Section III. Thereafter, Section IV describes the experimental evaluation of offloading context processing from the device to the Cloud service and discusses the experimental results of varying scenarios, and the significance and impact is also provided. We conclude this article in Section V, which also elaborates the general future direction and specific targets that are planned for further exploration of the issues discussed in this article.

II. Background and Related Work

Badidi and Esmahi [5] have proposed a Cloud-based context provisioning system that focuses on quality of service in the provisioning of contextual information. However, their emphasis is on using Cloud-based decision making to select appropriate context services for context subscribers, and their system does not cater for the cost of processing complex contextual information for context consumers. Xiao et al. [6] have developed the CasCap framework, which utilizes a Cloud-based function for offloading and adaptable services to improve energy efficiency in data communication. However, CasCap is primarily a power management framework that uses the device context information and collaborates with Cloud services to provide efficient power management for mobile devices. Korpipää et al. [7] have specified some basic requirements for designing a contextual model in terms of simplicity (for easy manipulation and reasoning), flexibility (modification of existing concepts), generality (range of concepts that can be modeled) and expressiveness (encompassing the properties of the modeled concepts). XML and Ontology based context modeling and representation schemes generally provide adequate coverage of these properties, with each scheme having scenario and usage dependent strengths and weaknesses e.g. XML Schema based approaches are inherently simpler and lightweight in comparison to ontology based approaches but lack the generality offered by ontologies in terms of modeling a wider spectrum of domain knowledge. In this article, we will consider the processing of contextual information represented using XML based schemas.

There are two predominant methods of processing XML documents, Simple API for XML (SAX) and Document Object Model (DOM) [8]. In the SAX approach, the parser starts at the beginning of the source document and passes each piece of the document to the application in the sequence it finds it. The application receives the parsed data sequentially and it is not saved in memory. Hence any in-memory manipulation of the data is not possible e.g. data cannot be updated in memory and committed to the XML document. A parser using the DOM approach creates a tree of objects that represents the content and organization of data in the document. The parsed tree exists in memory but in this case the application can navigate through the whole tree to access the data it needs, and if appropriate, manipulate it. SAX parsers have certain benefits over DOM-style parsers. Primarily, the amount of memory that a SAX parser must use in order to function is typically much smaller than that of a DOM parser. This is so because DOM parsers must have

the entire tree in memory before any processing can begin, so the amount of memory used by a DOM parser depends entirely on the size of the input data. The memory footprint of a SAX parser, by contrast, is based only on the maximum depth of the XML file (the maximum depth of the XML tree) and the maximum data stored in XML attributes on a single XML element. Both of these are always smaller than the size of the parsed tree itself. Moreover, because of the event-driven nature of SAX, processing documents is often faster than DOM-style parsers. Memory allocation takes time, so the larger memory footprint of the DOM is also a performance issue. However, there are situations where DOM-style parsers are comparatively more useful, e.g. some kinds of XML processing simply require having access to the entire document e.g. XSLT and XPath based processing need to be able to access any node at any time in the parsed XML tree. This aspect of the DOM approach makes it more relevant to situations where the processing is to be carried out based on the whole content of the document rather than on a subset of elements e.g. XPath queries in ContextML based documents. XPath [9], proposed by the World Wide Web Consortium as a practical language for selecting nodes from XML document trees, is an XML query language. The role of XPath is at the core of several other XML-related technologies, such as XSLT, XPointer, and XQuery. However, a drawback of XPath based query processing is that it leads to an exponential time complexity [2], [10]. The complexity of an XPath query is largely dependent on the predicates and assertions specified in the query e.g. attributes and elements to match, and the size/depth of the document being processed. Efficient implementations of XPath processing have been proposed in literature that achieve to polynomial (combined) complexity [2]. However, such improved implementations have not been integrated in widely deployed XPath engines such as those available in Java, Android and .NET platforms.

A consequence of the complexity of XPath processing is that it is costly, in terms of time and space, to process large documents and/or complex queries, especially on mobile devices. Context data has a high degree of temporal significance; a context datum may be valid at one point in time and invalid at the next. Processing delays may invalidate the integrity of processed information. Hence, it is important to process contextual data as soon as possible and within the temporal bounds of the data itself. The computational complexity of executing XPath queries on mobile devices also affects the limited energy resources. While processing and storage capacity of mobile devices has continually increased over the last decade, energy capacity has not followed a similar trend and remains the limiting factor in the usability of smart mobile devices as a dependable computational platform. The significance of the limiting factor is evident from the hardware and software mechanisms put in place by the device manufacturers and platform developers for the sole purpose of energy conservation. These include dynamic CPU scheduling, dimming of displays during non-use scenarios, adaptable radio transmission power, CPU-wake lock based application execution etc. This issue is one of the main thrusts of this paper and we aim to identify the XPath query processing and offloading scenarios that lead to energy conservation on mobile devices by reduction of the processing burden. Before describing the experimental framework to analyze the XPath query processing costs, in terms of time and energy consumption, it is necessary to describe our federated broker based context provisioning system, titled the Context Provisioning Architecture, and the ContextML scheme.

III. Context Provisioning Architecture

A. Consumer-Broker-Provider Model

The Context Provisioning Architecture is based on the producer (provider)-consumer model in which context related services take the roles of context providers or context consumers. These basic entities are interconnected by means of context brokers that provide routing, event management, query resolution and lookup services. The following paragraphs describe these three main components of the architecture.

Context Consumer - A Context Consumer (CxC) is a component (e.g. a context based application) that uses context data. A CxC can retrieve context information by sending a subscription to the Context Broker (CxB) or a direct on-demand query and context information is delivered when and if it is available.

Context Provider - The Context Provider (CxP) component provides contextual information. A CxP gathers data from a collection of sensors, network/Cloud services or other relevant sources. A CxP may use various aggregation and reasoning mechanisms to infer context from raw sensor, network or other source data. A CxP provides context data only to a specific invocation or subscription and is usually specialized in a particular context domain (e.g. location).

Context Broker - A Context Broker (CxB) is the main coordinating component of the architecture. It works as a facilitator between other architectural components. Primarily the CxB has to control context flow among all attached components, which it achieves by allowing CxCs to subscribe to context information and CxPs to deliver notifications.

A depiction of the core system components described above is presented in Figure 1, emphasizing the complementary provision of synchronous and asynchronous context-related communication facilities. A number of useful applications have been developed based on this architecture. Further details of this architecture and industrial trials are described in [11], [12].

Context consumers and providers register with a broker by specifying its communication end point and the type of context they provide or require. This in turn enables a

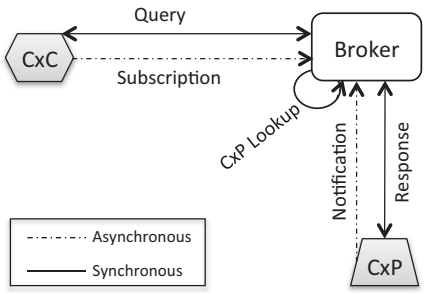


Figure 1. Basic broker based context provisioning component interaction

brokering function in which the context broker can lookup a particular context provider that a context consumer 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, as done routinely in internet communications to improve overall performance. A distinguishing feature of this architecture is the federation of multiple context brokers to form an overlay network of brokers Figure 2, which improves scalability of the overall system and provides location transparency to the local clients (CxCs and CxPs) of each broker. This federation of context brokers is achieved with a coordination model that is based on routing of context queries/subscriptions and responses/notifications across distributed brokers, discovery and lookup functions and is described in detail in our earlier work [4].

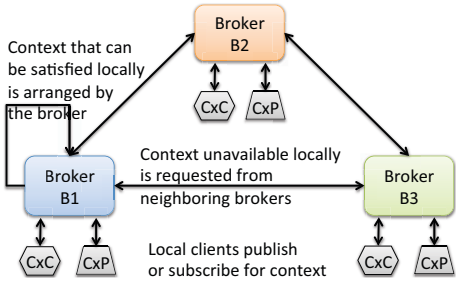


Figure 2. Simplified view of the federated broker based interaction

B. ContextML

ContextML is an XML based schema for the representation of contextual information. The defining principle in ContextML is that context data relates to an entity and is of a certain scope. The entity may be a user, a username, a SIP or email address etc., and scope signifies the type of context data e.g. weather, location, activity and user preferences.

The actual context information about scope is encoded using named parameters, parameter arrays and complex parameter structures in ContextML elements. In addition to the representation of contextual data, ContextML also contains a specification for control messages between components, subscriptions and notifications, component advertisements and routing related messages that are utilized in the overall system for coordination of context exchange. A parser, titled the ContextML Parser, has been implemented as a Java library for Java SE, EE and the Android platforms that can be used by context producing and consuming applications for the processing of contextual information and other messages encoded in ContextML. The model of the contextual data-related elements is depicted in Figure 3, detailed discussion about various dimensions of ContextML is presented in earlier work [3].

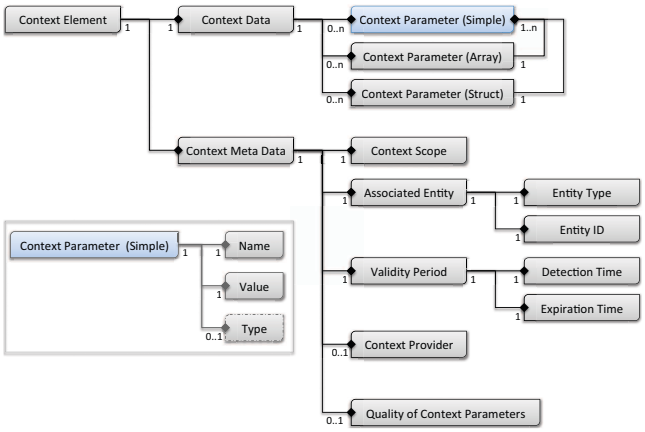


Figure 3. ContextML model of the `ctxEl` element

C. Context Processing Service

As discussed above, CxCs request context by subscribing with their CxBs. The CxBs look up the relevant CxPs and forward the subscriptions to the CxPs, which reply with the context data satisfying the particular context subscription. The CxBs maintain subscription tables and hence can forward incoming notifications to the subscribing consumers. A context consuming application usually subscribes to multiple context scopes in our architecture and has to infer complex context by processing multiple context messages. Keeping in mind the increasingly involved role of mobile devices in human-computer interaction and ubiquitous computing, the Context Provisioning Architecture allows the context consuming applications on mobile devices to submit XPath queries to be applied on context notifications generated in response to their subscriptions as well. Furthermore, the context consumers can submit contextual data and processing queries to the context brokers in order to decrease their processing burden. The context brokers provide these facilities through a Context Processing Service (CPS), which exposes RESTful HTTP based interfaces for CxCs to submit queries for processing over contextual data. The query and context data can be provided by the CxCs during invocation of the CPS or via the CxB (cp. Figure 4). Upon completion of query processing, the CPS returns the results to the

submitting client. It is expected that this model of offloading complex processing tasks should be beneficial in terms of time and space complexity on mobile devices. Consequently, such a mechanism could also affect the energy consumption on mobile devices and result in conservation of this critical resource.

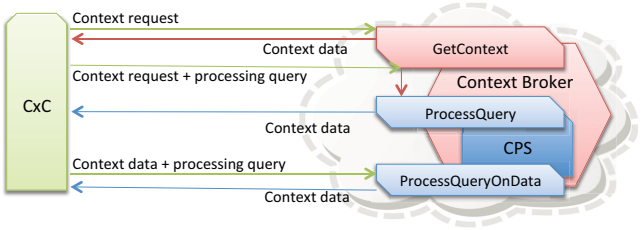


Figure 4. RESTful HTTP interfaces exposed by the CxB and CPS

Despite the simplistic offloading mechanisms obvious benefits, there are critical questions to be addressed. Firstly, the trade-off between the efficiency improved by offloading complex tasks, and that compensated by addition of the remote communication process, is not yet evident and quantified. Secondly, all contextual processing is not complex. The processing complexity of XPath, being used in our analysis, is dependent both on the size of the document (ContextML based information) and the complexity (number of parameters) of the XPath expression/query. The cost of processing of simple XPath expressions on small sized ContextML documents may be less than that incurred when offloading to a remote Cloud based service (due to addition of the communication cost incurred during data transfer from device to Cloud service). The size of the ContextML document, the complexity of the XPath query, the processing time, the communication cost and energy consumption are the main parameters that need to be evaluated in order to establish the realistic benefits of offloading complex query processing from mobile devices to the Context Processing Service. The following section describes our experimental analysis of these factors and its results in detail.

IV. Context Processing Offloading Analysis

A. Experimental Framework

Figure 5 depicts the system architecture of the Context Provisioning Architecture relevant to the focus of this discussion. The experimental framework consists of a CxB deployed on a network host and a CxC deployed on an Android based mobile device (Google Nexus One). The device and the CxB are connected through an IEEE 802.11g based WLAN setup. The CxB exposes RESTful HTTP based interfaces to the CxC for requesting context from CxPs and submitting contextual queries for processing to the CPS. Context requests, queries and responses are exchanged between the interacting components in the form of ContextML encoded messages over the RESTful HTTP

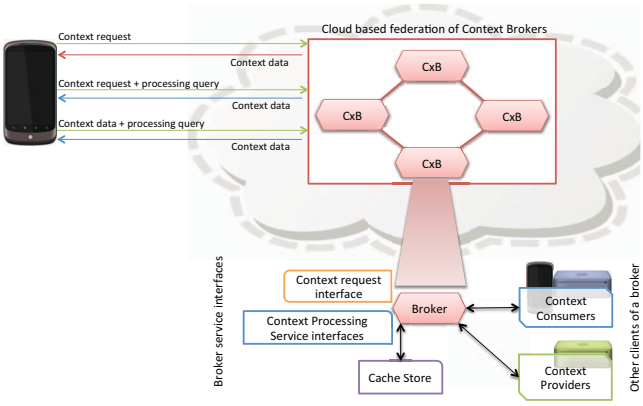
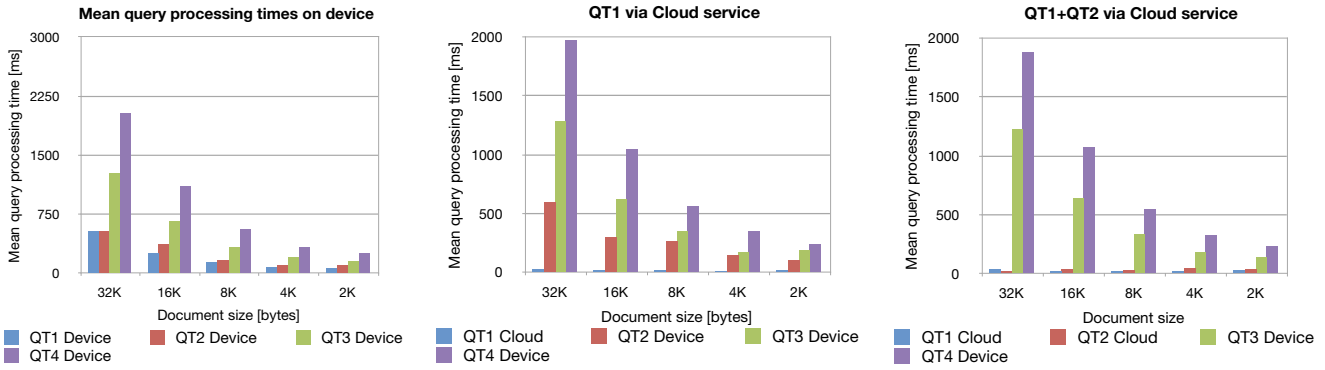


Figure 5. System architecture with focus on components relevant to the experimental framework

interfaces. In order to reduce the variable parameters from the experimental framework, CxPs are not utilized rather the CxB is provided with stored context notifications messages spanning 10 context scopes and 10 entities. The contextual data has been recorded from real-world deployments of various context providers of the Context Provisioning Architecture and include contextual scopes of user location (GPS), weather, proximity, activity, wi-fi (nearby access points), bluetooth (nearby devices), user profile, device status, device settings and device location (cellular). The document sizes of the ContextML encoded context information range from 2KB to 32KB; Figure 6 shows a ContextML snippet from one such document, containing the wf scope context that contains contextual information about detected/known Wi-Fi access points and their properties.

```
<contextML>
 <ctxEls>
  <ctxEl>
   <contextProvider id="DevCxP-354957030977071" v="0.8" />
   <entity id="354957030977071" type="imei" />
   <scope>wf</scope>
   <timestamp>...</timestamp>
   <expires>...</expires>
   <dataPart>
    <par n="wfList">...</par>
    <parA n="wfDevices">
     <parS n="wfDevice">
     <par n="wfName">SLK</par>
     <par n="wfBssid">0024B29A4D18</par>
     <par n="wfType">[WPA-PSK-TKIP]</par>
     <par n="wfSignal">-77</par>
     <par n="wfOpen">false</par>
     </parS>
    </parA>
    <parA n="wfDevices">
     <parS n="wfDevice">
      <par n="wfName">SLK</par>
      ...
      ...
     </parS>
    </parA>
   </dataPart>
  </ctxEl>
 </ctxEls>
</contextML>
```

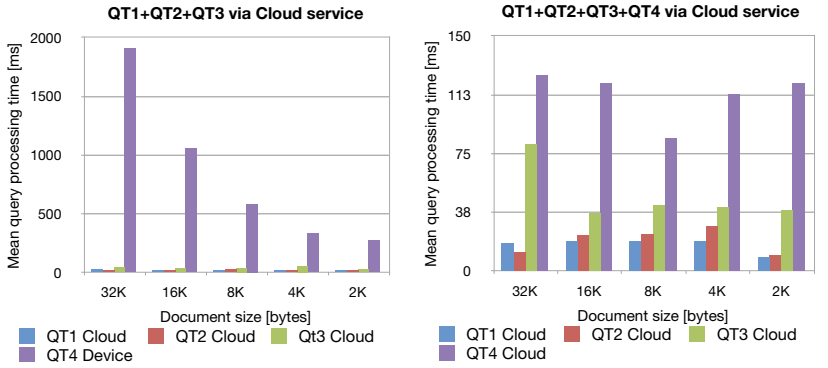
Figure 6. ContextML snippet from a document containing contextual information about the detected and known Wi-Fi access points in proximity to an entity.



(a) Mean query processing time – all queries processed at the device

(b) Mean query processing time – QT1 queries processed by the CPS, rest at the device

(c) Mean query processing time – QT1 and QT2 queries processed by the CPS



(d) Mean query processing time – QT1, QT2 and QT3 queries processed by the CPS

(e) Mean query processing time – all queries processed by the CPS

Figure 7. Query processing times – experimental results

The CxC on the mobile device is also provided with these ContextML documents for use when the cost of local device-based processing is to be evaluated. The CxC is also configured to issue context queries spanning these documents. The queries are divided into four categories of complexity, consisting of:

1) *QT1*, an XPath expression that searches for a fixed string value of parameter n in any element i.e. `//*[contains(@n,'string')]`
2) *QT2*, an XPath expression that searches for two different string values of any parameter in any element i.e. `//*[contains(.,'string)] | //*[contains(.,'string')]`
3) *QT3*, an XPath expression that searches for a particular combination of parameter n values along an element path /parS/par i.e. `//parS[contains(@n,'string')] /par[contains(@n,'string')]`
4) *QT4*, an XPath expression that retrieves all parameter *n* values from the document i.e. `//*[@n]`

B. Results

Figures 7b, 7c, 7d and 7e illustrate the reduction in mean query processing times when QT1, QT1 + QT2, QT1 + QT2 + QT3 and all types of query processing is offloaded to the CPS. The communication time between the device and the CPS is minimal as the experiments are performed in an isolated WLAN.

Figure 10 aggregates the various scenarios and demonstrates the gradual reduction in mean query processing times as subsequent query types are offloaded to the CPS. The graph shows that the relevant benefits, in terms of improved processing time, increase with the increase in the document size and the complexity of the queries. We can observe from Figure 10 that the there is a minor improvement in mean query processing time between the cases when QT1 is offloaded to the CPS and when QT1+QT2 are offloaded. However, the improvement is greater when complex query types (QT3 and QT4) are offloaded as well. These trends are visible from the slope of the lines representing each experiment set in Figure 8. Table I further quantifies the total time taken when a certain experiment set is carried out across all document sizes. The results show that offloading complex query types to the CPS provide greater improvements in the overall mean query processing time.

The reduction in processing burden also affects the energy consumption on the device. Figure 9 illustrates the energy consumption plot during different query processing scenar-

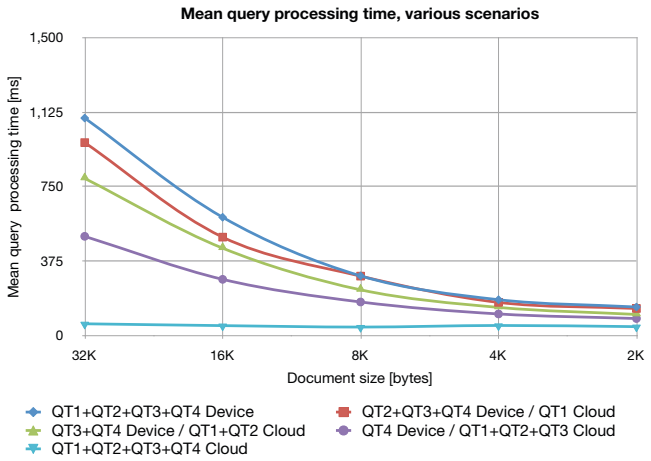


Figure 8. Mean query processing times – various scenarios

Table I
QUERY PROCESSING TIMES

Query processing location	Total processing time across all document sizes (2K–32K) [ms]
All QT on Device	2314
QT1 CPS	2068.25
QT1+QT2 CPS	1708.25
QT1+QT2+QT3 CPS	1144.75
QT1+QT2+QT3+QT4 CPS	246.25

ios. PowerTutor [13] is used for calculating the energy used by individual applications (broker, consumers and providers) on the device. PowerTutor is an application for Google phones that displays the power consumed by major system components such as CPU, network interface, display, etc. and different applications. This application allows software developers to see the impact of design changes on power efficiency. PowerTutor calculates the phones breakdown of power usage with an average of 1% error over 10-second intervals while the worst case error over 10 seconds is 2.5% [13, pg. 8]. In these experiments only the energy used by an application in utilizing the CPU and Wi-Fi is considered when calculating its energy consumption signature. Executing all query sets on the device consumes 131.4J in our experiment scenario. Offloading QT1 and QT2 to the CPS results in a reduction of about 11.5% energy cost at each step, whereas the offloading of QT1, QT2 and QT3 to the CPS results in 38.4% reduction in energy consumption when compared to offloading only QT1 and QT2 queries. Expectedly, offloading all queries to the CPS results in minimal energy consumption at the device: 0.74J in our experiments. These relative energy consumption measurements follow a similar trend as in case of mean query processing time (Figure 8, Table 1I) i.e. energy consumption improves when complex query types are offloaded to the CPS. These results signify the benefit of our approach in terms of energy conservation on mobile devices taking part in context consumption and processing. Moreover, it highlights the beneficial and supportive role context-aware Clouds can play in processing and distribution of contextual information.

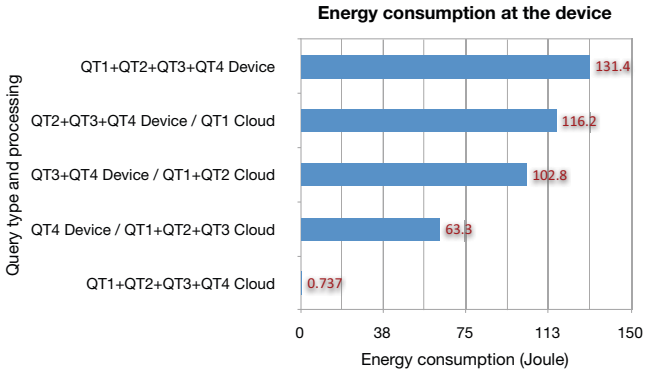


Figure 9. Energy consumption at the device – various scenarios

V. CONCLUSION AND FUTURE WORK

This article has briefly described the Context Provisioning Architecture, focusing on analysis of ContextML query processing, on a smart mobile device, and experimentally evaluating the benefits of offloading complex query processing to a Context Processing Service. The RESTful HTTP based access of the CPS and the CxBs can be ideally utilized by deployment in a Cloud infrastructure to exploit the benefits of scalability in increasing load scenarios, providing device independence, and improving reliability. Our experimental framework has considered XPath based queries in four categories of complexity and demonstrated the computational cost associated with such processing on a mobile device. The experiments evaluated the improvements in query processing times, as the processing of queries in various complexity categories is offloaded to the CPS. The results have demonstrated significant improvement in the overall query processing times. This improvement has a direct effect on the energy consumption of the mobile device during the query processing as well.

The distribution of queries is uniform in our experimental framework, i.e. the queries in each complexity category are equally distributed in the experiments. Furthermore the distribution of queries over the size of ContextML documents is also equally distributed. While this setup allows for a basic assessment of query processing costs, it may not occur in real world scenarios too often. Therefore, we plan to repeat the experiments with a non-uniform distribution of queries and ContextML document sizes. Specifically, some types of contextual information (ContextML scopes) are more frequently accessed than others, and hence the distribution of queries over the size of ContextML documents also

becomes non-uniform. Furthermore, we have only considered communication through the Wi-Fi interface of the device, whereas a real-world scenario may involve a greater portion of communication being carried through the 3G interface on the device, which has different speed and energy consumption characteristics than the Wi-Fi interface. This factor may influence the energy consumption and improved processing time related benefits that we have quantified in this paper. These issues are the main focus of our future work.

REFERENCES

[1] S. L. Kiani, B. Moltchanov, M. Knappmeyer, and N. Baker, "Large-scale context-aware system in smart spaces: Issues and challenges," in *Baltic Congress on Future Internet and Communications – Special Session on Smart Spaces and Ubiquitous Solutions*, Riga, Latvia, February 2011.

[2] G. Gottlob, C. Koch, and R. Pichler, "Efficient algorithms for processing XPath queries," *ACM Transactions on Database Systems*, vol. 30, no. 2, pp. 444–491, 2005.

[3] 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, pp. 367 – 372.

[4] S. L. Kiani, M. Knappmeyery, N. Baker, and B. Moltchanov, "A federated broker architecture for large scale context dissemination," in *Proceedings of the International Conference on Computer and Information Technology*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 2964–2969.

[5] E. Badidi and L. Esmahi, "A cloud-based approach for context information provisioning," *World of Computer Science and Information Technology Journal*, vol. 1, no. 3, pp. 63–70, 2011.

[6] Y. Xiao, P. Hui, P. Savolainen, and A. Ylä-Jääski, "Cascap: cloud-assisted context-aware power management for mobile devices," in *Proceedings of the second international workshop on Mobile cloud computing and services*, ser. MCS '11. New York, NY, USA: ACM, 2011, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/1999732.1999736

[7] P. Korpipää, J. Kela, and E. J. Malm, "Managing context information in mobile devices," *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 42–51, 2003.

[8] E. R. Harold, *Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.

[9] J. Clark and S. DeRose, "XML path language (XPath) version 1.0," *W3C Recommendation*, vol. 16, 1999.

[10] P. Wadler, "Two semantics for XPath," Bell Labs, Tech. Rep., 2000.

[11] M. Zafar, N. Baker, B. Moltchanov, S. L. João Miguel Goncalves, and M. Knappmeyer, "Context management architecture for future internet services," in *ICT Mobile Summit 2009*, Santander, Spain, June 2009.

[12] M. Knappmeyer, R. Tönjes, and N. Baker, "Modular and extendible context provisioning for evolving mobile applications and services," 2009.

[13] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 105–114. [Online]. Available: http://doi.acm.org/10.1145/1878961.1878982