Sixth Annual IEEE International Conference on Pervasive Computing and Communications

Peer-to-Peer Context Reasoning in Pervasive Computing Environments

Tao Gu [1], Hung Keng Pung [2], Daqing Zhang [1]

[1] Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore

[2] National University of Singapore, 3 Science Drive 2, Singapore

tgu@i2r.a-star.edu.sg; punghk@comp.nus.edu.sg; daqing@i2r.a-star.edu.sg

Abstract

In this paper, we propose a peer-to-peer approach to derive and obtain additional context data from low-level context data that may be spread over multiple domains in pervasive computing environments. In this system, peers are self-organized into a semantic peer-to-peer network as the underlying communication substrate. Context reasoning is done in a distributed fashion through logical reasoning according to a set of user-defined rules. Both pull and push services are supported in the system to enable message exchange during the reasoning process. We present our design concepts, and prove the effectiveness of our system through the prototype evaluation.

1. Introduction

The advancement of pervasive computing advocates complex applications to be built with context-awareness enabled. In the past decade, there have been many efforts on building various context-aware systems. One of the challenges in these systems is how to derive high-level context data from a set of low-level context data that may be spread over multiple domains. In recently years, many systems such as [2] adopted a centralized approach to context reasoning. This approach works well in a single smart space, i.e., a smart home. It can provide fast response, and it is relatively easy to update context data or indices since the data set in a single domain is small. However, the centralized approach has traditional limitations such as a single processing bottleneck and a single point of failure. It requires system administration at the centralized server. More importantly, it may not be suitable for building cross-domain context-aware applications that require utilizing and reasoning about context data over multiple smart spaces. Using a centralized approach for such applications may lead to increased difficulties in administration and a higher maintenance cost of updating context data due to the dynamicity of context information.

Recently, peer-to-peer (P2P) approaches such as [6], [7] and [8] have received considerable attention and gained popularity because their underlying infrastructures are appropriate to scalable and flexible distributed applications. In a P2P system, there is no centralized control: Each peer acts as a server or a client, and cooperates with other peers in order to solve a collective task. Performing context reasoning in a P2P manner provides us a potential solution to overcome the limitations incurred by a centralized reasoning system.

In this paper, we propose a P2P approach to context reasoning in pervasive computing environments. We aim at providing reasoning capabilities for collaborative context-aware applications over multiple domains. In this system, peers are organized into a semantic P2P network [9]. Context interpreters are special peers that are responsible for performing reasoning tasks through logical reasoning which consists of ontology reasoning and rule-based reasoning. We extend the semantic P2P network by proposing an event subscription mechanism that allows applications to pre-subscribe context data to keep track of dynamic data changes in the reasoning process. We conduct the prototype evaluation to prove that our P2P reasoning system works effectively.

The rest of the paper is organized as follows. We survey related work in Section 2, and then present the architecture of our P2P context reasoning system in Section 3. We present the evaluation results in Section 4. Finally, we conclude our work in Section 5.

2. Related Work

Many existing systems adopted a centralized approach to derive high-level contexts. Chen et al. [2] proposed the CoBrA infrastructure where context information is shared by all devices, services and agents. CoBrA deploys a centralized server called Context Broker to store and reason about context

0-7695-3113-X/08 $25.00 © 2008 IEEE
DOI 10.1109/PERCOM.2008.37

IEEE computer society

Authorized licensed use limited to: RMIT University Library. Downloaded on January 10,2021 at 16:50:47 UTC from IEEE Xplore. Restrictions apply.

information. Wang et al. [10] proposed a centralized context reasoning system to acquire and aggregate low-level contexts, and derive high-level contexts. However, the centralized approach has limitations such as a single processing bottleneck, which ultimately leads to a scalability problem; and a single point of failure, which undermines system robustness. This approach also requires system administration at the centralized server. Studies in [2] and [10] have shown that logical reasoning is a computationally intensive process, and a centralized reasoning engine may not scale well when the knowledge base and/or the rule set are large.

Researchers also proposed many context reasoning systems based on distributed reasoning servers. Ranganathan et al. [1] developed a middleware infrastructure to enable context awareness in pervasive computing. They leverage on standard CORBA services to manage and resolve the basic difficulties in building a distributed context reasoning system. Our previous work in [3] proposed a distributed context reasoning system based on Service-Oriented Architecture. This approach distributes the workload of a centralized reasoning server to multiple servers in the network; however, distributed servers still have the limitations of robustness and centralized administration, and need an advanced load-balancing strategy.

Przybilski [5] proposed a distributed approach using P2P for context reasoning. In this system, mobile devices perform simple context reasoning such as feature extraction, and can send their reasoned context information to a remote server for more advanced context reasoning such as classification that requires more powerful computation capabilities. While this concept is new, the feasibility of this approach remains unproved as this system is neither implemented nor evaluated.

3. System Architecture

Our system consists of many individual peers, which can act as context producers, context consumers or context interpreters. A context producer provides various context data – usually low-level context data that are obtained from physical sensors, whereas a context consumer obtains context data by sending context queries to the network and receiving results. A context interpreter is a special context producer which is able to derived implicit and high-level contexts from explicit and low-level contexts. We organize context producers and interpreters into a P2P network [9] – an ontology-based semantic network as the underlying communication substrate. A context consumer submits its context queries to any context producer or interpreter that acts as a proxy, and receives the results. Users or applications may utilize various contexts obtained from the network and adapt their behaviors accordingly.

3.1. Data Model and Storage

In our system, we use an RDF based context model to represent context data. We have designed our context ontologies with a two-level hierarchy in [3]. The upper ontology defines common concepts about the physical world in pervasive computing environments, and it is shared among all the peers. Each peer can define its own concepts in its low-layer ontologies which extend the leaf concepts in the upper ontology. Different peers may store different sets of low-layer ontologies based on their applications' needs. This design approach offers application developers the flexibility to define the domain knowledge which is specific to their applications.

Each context producer or interpreter maintains a local RDF repository to store local context data and ontologies. An RDF repository can be accessed through the RDQL based query engine which is able to parse RDF triples and resolve context queries.

3.2. First-Order-Logic

We use first-order-logic to reason about context data. There are two kinds of reasoning in our system: ontology reasoning and user-defined rule-based reasoning. Ontology reasoning is responsible for checking class consistency and implied relationship, asserting inter-ontology relations when integrating or switching domain-specific ontologies.

Application developers can also define their own rules to derive high-level contexts. This enables users or applications to raise the level of context abstraction based on their needs. A user-defined rule takes the form:

```
<RuleName>: <Premise1> ... <Premisen> -> <Conclusion>
```

`<RuleName>` specifies the name of the rule. `<Premise1>...<Premisen>` are triple patterns representing the premises that make the conclusion true. `<Conclusion>` is a triple pattern that specifies the high-level statement generated when all the premises are satisfied.

Each context interpreter stores a set of rules locally. For context queries, the rules to infer high-level contexts are encoded as backward chaining rules. Given a high-level context query, they help identifying and invoking one or more low-level context data. For

Authorized licensed use limited to: RMIT University Library. Downloaded on January 10,2021 at 16:50:47 UTC from IEEE Xplore. Restrictions apply.

context subscription, we use forward chaining reasoning to infer the facts when context changes. This is to avoid having to infer the same facts repeatedly. In the implementation of rule-based reasoning engine, we combine both forward chaining and backward chaining models to form a hybrid execution mode.

3.3 Semantic P2P Overlay and its Extension of Event Subscription

In our system, context producers and interpreters are organized into a semantic P2P network. We group context producers and interpreters into different semantic clusters. Each semantic cluster corresponds to a leaf concept in the upper context ontology. Upon creation, each context producer or interpreter first maps the semantics of its local data to one or more leaf concepts in the upper context ontology; it then joins a semantic cluster. These semantic clusters are interconnected as a one-dimensional circular space to form the P2P network. Upon receiving a context query, the context producer or interpreter pre-processes it to obtain the semantic cluster associated with the query, and then routes the query to appropriate producers or interpreters that have the relevant context data.

In addition to search requests which pull data from the network on a one-time basis, context consumers may be interested to subscribe a context event to the network and get notified when the event changes over a period of time. In this paper, we propose an event subscription mechanism which is an extension to the semantic P2P network. It is particularly useful during the reasoning process where one or more premises are not ready at the point of a derivation; and hence, they need to be subscribed to the network and resume the derivation process when all premises get ready.

When a context producer or interpreter receives a subscription request, it checks its local RDF data and decides whether it should accept the request. For example, a peer subscribes a context event "John enters the bedroom" in the RDF triple form of `<socam:John socam:locatedIn socam:Bedroom>` to the network. As this RDF triple may not exist in the network (i.e., John may be in some other places) at the time of receiving a request, the subscription request may end up with no context producers to register. To avoid losing potential producers or ending up with many irrelevant producers, we propose a subscription acceptance algorithm, as shown in Figure 1, for a peer to match a subscription request against its local RDF data.

```
Given a subscription request in the form
of an RDF triple pattern <Subs, Preds,
Objs>, a variable in the RDF triple
represents any arbitrary constant.
Let <Subl, Predl, Objl> represents any RDF
triple in a peer's local data set called
L.
accept = false;
for each RDF triple in L
  if Preds is of DatatypeProperty && ((Subs
== Subl) ∩ (Preds == Predl)) == true then
    accept = true; break;
  else if Preds is of ObjectProperty
&&((Preds == Predl) ∩ (Objs == Objl)) ==
true then
    accept = true; break;
  end if
end for
if accept == true then accept the request;
else reject the request;
end if
```

Figure 1: Subscription acceptance algorithm

This algorithm works for a subscription request in the form of any RDF triple pattern whose subject, predicate or object may take variables. Although predicates can be specified as variables, this situation seldom occurs since users or applications are always in favor of more specific events in context-aware pervasive applications. We now consider the case that a predicate is specified in a subscription request. If a subscription request's predicate is of type `DatatypeProperty`, a context producer determines if its local RDF data contains triple(s) with the same subject-predicate pair as in the request. For example, for a given subscription request `<socam:Bedroom socam:lightLevel 'LOW'>`, a producer will accept the request if there exists an RDF triple with subject `socam:Bedroom` and predicate `socam:lightLevel` in its local data. If a subscription request's predicate is of type `ObjectProperty`, a context producer determines if its local RDF data contains triple(s) with the same predicate-object pair as the request. For example, for a given subscription request `<socam:John socam:locatedIn socam:Bedroom>`, a producer will accept the request if there exists an RDF triple with predicate `socam:locatedIn` and object `socam:Bedroom` in its local data.

To understand the rationale behind these decisions, consider a subscription request in the form of the RDF triple `<Subs, Preds, Objs>`. Such a triple may be obtained from raw data generated by a physical sensor. In the sensor network domain, a predicate always corresponds to a sensor type, e.g., `socam:locatedIn` corresponds to a physical

Authorized licensed use limited to: RMIT University Library. Downloaded on January 10,2021 at 16:50:47 UTC from IEEE Xplore. Restrictions apply.

location sensor. If $Pred_S$ is of `DatatypeProperty`, Sub_S should correspond to the target this sensor is monitoring and Obj_S should correspond to the sensor output. For example, the RDF triple of <socam:Bedroom socam:lightLevel 'LOW'> can be interpreted as the output of a light level sensor monitoring the bedroom's light level. If a context producer's local RDF data contains at least one triple with the Sub_S-$Pred_S$ pair, it can be inferred that this producer has the type of sensor specified by this pair. Hence, we can conclude that this producer can provide triples of this same subject-predicate pair. On the other hand, if $Pred_S$ is of `ObjectProperty`, Obj_S should correspond to the target this sensor is monitoring and Sub_S should correspond to the sensor output. In this case, the producer can provide triples with the same Sub_S-$Pred_S$ pair as in the subscription request.

Once a peer accepts a subscription request, it keeps monitoring the request. Whenever a change occurs, the peer notifies the subscribers if the RDF triple matches the subscription request. An RDF triple $<Sub_C\ Pred_C\ Obj_C>$ is said to match the subscription request if $(Sub_C == Sub_S) \cap (Pred_C == Pred_S) \cap (Obj_C == Obj_S)$ are true. The routing of notification traces the exact path of the subscription request in the reverse direction. A subscriber can unsubscribe a context event by sending an unsubscription request directly to the producer.

4. Evaluation Results

To evaluate our prototype system, we set up a testbed that consists of eight peers (five producer peers and three interpreter peers) in our campus network. We run all the peers on off-the-shelf computers – Pentium 500MHz desktop PCs with 256MB memory to see how well the system performs. We create and store a set of low-level context data in each context producer, and create different sets of rules in the three interpreters for deriving high-level contexts. All the peers participate in the overlay network. We evaluate our prototype system based on the testbed and focus our evaluations on the performance of context interpreter.

4.1. Deduced Query Processing Capability

In this experiment, we evaluate the capability of a context interpreter to process deduced queries. We generate a various number of simultaneous deduced queries in randomly selected peers, and measure the average processing time in the context interpreters. Figure 2 plots average processing time against number of simultaneous deduced queries. When a logarithmic scale is used for both axes, the graph displays a linear relationship. This shows that the query processing capability of context interpreter scales well to number of deduced queries.

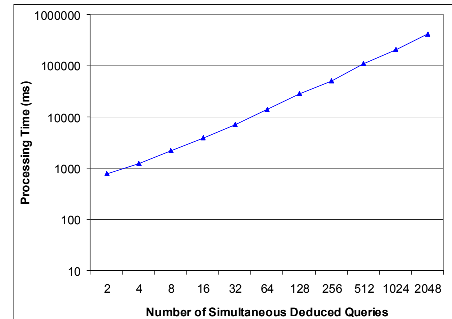

Figure 2: Query processing capability of context interpreter.

On average, a context interpreter takes about 1.1 seconds to answer a deduced query. We analyze query response time by breaking it down into three portions: query mapping, query processing and communication, as shown in Figure 3. Query mapping is the time taken to map a query to the appropriate cluster for routing. Query processing is the time taken to process a deduced query, including tasks such as rule processing, internal query generation, and high-level context data derivation. Communication represents the time taken for queries and their results to travel over the network. It includes communication cost for both deduced queries and internal queries.

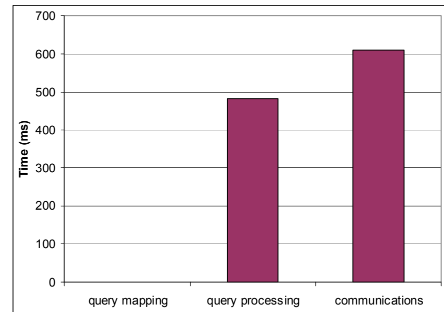

Figure 3: Breakdowns of the response time for deduced query.

As we can see from the above figure, the costs of query processing and communication are the major factors. We have assumed the worst scenario in this experiment, i.e., a context interpreter's premise model is empty and all the premises have to be obtained from the network by issuing internal queries. In the reminder of this section, we will look into various methods to improve deduced query processing.

Authorized licensed use limited to: RMIT University Library. Downloaded on January 10,2021 at 16:50:47 UTC from IEEE Xplore. Restrictions apply.

4.2. Improving Deduced Query Processing

We propose and evaluate four methods to improve deduced query processing for context interpreter. Table 1 summarizes the four methods.

TABLE 1: DIFFERENT METHODS FOR DEDUCED QUERY PROCESSING

Method	Pre-subscription	Shared Internal Subscription
A	✓	×
B	×	×
C	×	✓
D	✓	✓

- **Method A** (Pre-subscribe all premises): The context interpreter analyzes all the rules it maintains and sends internal queries to the network for all possible premises upon startup. It also pre-derives all possible high-level context data corresponding to all the rules it stores.
- **Method B** (Internal queries are not shared): For each deduced query received, a context interpreter sends internal queries for all relevant premises and unsubscribes these internal queries when the deduced query is answered. Internal queries are not shared between rules.
- **Method C** (Internal queries are shared): This method is similar to Method B, but internal queries are shared between rules. A context interpreter only sends an internal query if it has not been sent and only unsubscribes an internal query if there are no pending deduced queries that require that internal query.
- **Method D** (Pre-subscribe certain premises/Internal queries are shared): This method is a combination of Methods A and C. A context interpreter uses Method A for the rules corresponding to frequent deduced queries and uses Method C for the rules corresponding to infrequent deduced queries.

In this experiment, we evaluate the effectiveness of each method. Method A is performed by manually placing the necessary high-level statements in a context interpreter's inference model beforehand. Method C is performed by checking for internal queries that have already been sent, and only initiating internal queries if they have not been sent before. Method B is performed by removing the check for internal queries that have already been sent. Hence, duplicated internal queries may be sent over the network. Method D is performed by manually placing a portion of all possible high-level statements in a context interpreter's inference model beforehand. Note that we use one-third of the high-level statements in this experiment, the ratio should be computed based on query statistics in real scenarios.

Figure 4 plots processing time of deduced queries for different methods. Among the four methods, Method A gives the shortest response time. This is because the context interpreter processes reasoning rules, subscribes internal queries and derives high-level context data beforehand. Method B performs the worst as we assume the worst-case scenario (i.e., a context interpreter has to issue an independent internal query for each of the premises).

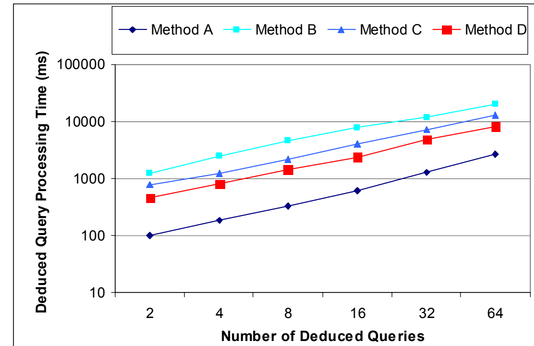

Figure 4: Processing time of deduced queries for different methods in the context interpreter.

Although Method A performs the best, it may not scale well if a context interpreter maintains too many rules. This is because the context interpreter would have to maintain a large number of internal subscriptions at all times. In addition, many irrelevant internal queries may be sent to the network, thus increasing unnecessary load on the network. Also, the context interpreter may have to periodically resend all internal queries to ensure that it is able to obtain premises from context producers that have just joined the network. Method B removes the need for a context interpreter to keep track of the internal queries it sends to the network and ensures that the premises obtained are fresh as internal queries are only sent when receiving deduced queries. However, this method is inefficient as it generates many redundant internal queries, which increase the network load and the response time for deduced queries. Method C provides a good compromise between Methods A and B. Although its response time is greater than that of Method A, it generates a significantly smaller number of internal queries compared to both Methods A and B, and thus reduces the network load. In addition, the premises obtained are also fresh as the internal queries are subscribed on demand as it is with Method B. Method D improves the response time of Method B by pre-subscribing the premises for deduced queries that are popular. It requires a context interpreter to maintain and keep track of the statistics of deduced queries received and deploy an algorithm to decide which

Authorized licensed use limited to: RMIT University Library. Downloaded on January 10,2021 at 16:50:47 UTC from IEEE Xplore. Restrictions apply.

queries should be pre-subscribed beforehand. This method needs to be further studied in our future work.

4.3 Memory Consumption

We have evaluated and analyzed the communication costs of various methods for handling deduced queries in Section 4.2. We now evaluate memory consumption. Figure 5 plots the memory consumption in term of MB (megabytes) for the above four methods.

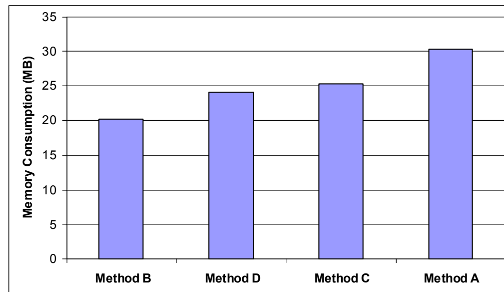

Figure 5: Memory consumption for the four methods.

Among them, Method A consumes the most memory. This is because, in Method A, a context interpreter has to maintain internal subscriptions for all internal queries to which it has pre-subscribed. Method B consumes the least memory because a context interpreter does not need to maintain internal queries as all internal queries are subscribed on demand. The memory consumptions of Methods C and D fall between that of Methods A and B. Clearly, there is a tradeoff between query response time and memory consumption. This evaluation also reveals that the computing device that runs context interpreter does require certain hardware capabilities (i.e., processing power and memory) apart from certain software platform capabilities. Some embedded computing device may not be capable of running the context interpreter, for example, mobile phone, etc.

5. Conclusion

In this paper, we have presented a peer-to-peer approach to derive high-level contexts from a set of low-level contexts that may be spread over multiple domains in pervasive computing environments. Our evaluation results have shown the effectiveness of our prototype system. We plan to develop several cross-domain context-aware applications (e.g., cooperative smart home application, etc) that fully utilize the capability of context interpreter in our future work.

10. References

[1] Anand Ranganathan and Roy H. Campbell. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In Proceedings of ACM/IFIP/USENIX International Middleware Conference, Brazil, June 2003.

[2] Harry Chen, Tim Finin, Anupam Joshi. Semantic Web in in the Context Broker Architecture. In Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communication, Orlando FL., March 2004.

[3] Tao Gu, Hung Keng Pung, Daqing Zhang. A Service-Oriented Middleware for Building Context-Aware Services. Elsevier Journal of Network and Computer Applications (JNCA), Vol. 28, Issue 1, pp. 1-18, January 2005.

[4] Jari Forstadius, Ora Lassila & Tapio Seppänen. RDF-Based Model for Context-Aware Reasoning in Rich Service Environment. In Proceedings of the 2nd Workshop on Context Modeling and Reasoning (CoMoRea) at 3rd IEEE International Conference on Pervasive Computing and Communication (PerCom'05), March 2005.

[5] Michael Przybilski, Petteri Nurmi, Patrik Floréen. A Framework for Context Reasoning Systems. Proceedings of the 23rd IASTED International Conference on Software Engineering, 2005.

[6] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In Proceedings of ACM SIGCOMM, 2001.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In Proceedings of ACM SIGCOMM, 2001.

[8] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale peer-to-peer systems. In Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms, Lecture Notes in Computer Science, November 2001.

[9] Tao Gu, Hung Keng Pung, Daqing Zhang. A Peer-to-Peer Overlay for Context Information Search. In Proceedings of the 14th International Conference on Computer Communications and Networks (ICCCN 2005), San Diego, California, October 2005.

[10] X. H.Wang, T. Gu, D. Q. Zhang, H. K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In Proceedings of Workshop on Context Modeling and Reasoning (CoMoRea 2004), In conjunction with the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), Orlando, Florida USA, March 2004.

Authorized licensed use limited to: RMIT University Library. Downloaded on January 10,2021 at 16:50:47 UTC from IEEE Xplore. Restrictions apply.