A System for Distributed Context Reasoning

Stamatia Rizou, Kai Häussermann, Frank Dürr, Nazario Cipriani, Kurt Rothermel
Universität Stuttgart, Institute of Parallel and Distributed Systems, Universitätstraße 38, 70569 Stuttgart, Germany
<firstname.lastname>@ipvs.uni-stuttgart.de

Abstract—Context aware systems use context information to adapt their behaviour accordingly. In order to derive high level context information from low level context, such as sensor values, context reasoning methods that correlate observable context information, are necessary. Several context reasoning mechanisms have been proposed in the literature. Usually these mechanisms are centralized, leading to suboptimal utilization of network resources and poor system performance in case of large-scale scenarios. Therefore to increase the scalability of context reasoning systems the development of methods that distribute the reasoning process is necessary. Existing distributed approaches are method specific and do not provide a generic formalization for distributed reasoning. In this paper we introduce a novel system which enables distributed context reasoning in a generic way that is independent of the reasoning algorithm.

I. INTRODUCTION

During the last years, the proliferation of modern devices, capable of capturing context information, has raised the interest of researchers to build context aware systems that use this information to adapt automatically their behaviour. However, low level context, which is directly acquired from sensors, is usually not attractive to the user as it refers to a partial perception of the events occurring in the environment. For instance, consider a person who is interested in using a room to make an unscheduled meeting. In such a case, the user would rather expect to be notified about the occurrence of a meeting, than about the light or the noise level in the room. In our system we refer to such high level context (e.g. "meeting in a room"), derived from the processing of observable context as situation. To recognize situations a reasoning mechanism is required.

Many of the existing context aware systems are designed to support specific use case scenarios (e.g. MS Easy Living [1] or Semantic Space [2]) and cover a limited geographical area (e.g. one building or conference room). Therefore they rely on centralized architectures for processing context data. However this centralized approach cannot provide a scalable solution in scenarios with a large number of geographically distributed context sources. Gu at al. [3] use multiple servers, each one responsible for a certain geographic region, to perform the context reasoning. Although this approach is a first step towards distributed context reasoning, it still executes each reasoning task centrally in a dedicated server. Therefore it leads to poor utilization of network resources and limited system performance. Other existing approaches that enable the distribution of the reasoning task [4], [5] are method specific, since they refer to a specific reasoning algorithm and they do not address the problem of distributed context reasoning as an optimization problem to achieve efficient utilization of network resources and high system performance. In this paper we introduce a system, which is based on a generic formalization that allows for the use of different reasoning algorithms and distribute the reasoning process according to different optimization criteria among a set of servers.

The motivation for the distribution of the reasoning process derives from the distributed nature of the data sources. Consider for instance a simple use case scenario where the system should detect the situation "traffic jam" along a user-defined route. Already for this simple scenario we need several pieces of low level context (e.g. distance between the cars, average speed), which could come from different sources (e.g. cars, road-side units, cameras on bridges). The question that naturally arises is at which server to correlate this distributed context data. One simple solution is to collect all the necessary information at a central server and perform the reasoning there. Obviously this approach does not use the network resources in an efficient way, as the data should be sent across the network without being filtered by the sources. In order to allow for the efficient utilization of network and processing resources and to improve the quality of the situation detection in terms of timeliness, mechanisms for distributing the reasoning process to a set of servers are required.

The main contribution of this paper is a system for enabling distributed context reasoning. Our approach is based on the idea of splitting the reasoning process up into a network of reasoning correlators that are distributed with respect to an optimization goal like minimal network usage or minimal delay. In detail, our system uses a situation-centric model, which contains predefined situation patterns, called situation templates that are stored as preknowledge in the system. Situation templates are built from observable context and processing units called operators. Different reasoning methods such as distributed Bayesian Networks can be supported through different operators implementing the specific context correlators. Generally each situation template forms a graph of operators, which cooperatively performs a reasoning task. At runtime, situation detection is initialized by the creation of a logical plan, which is derived from a situation template. This plan describes the detection of a concrete situation at a certain location or for a given object by an operator graph. Subsequently the system finds a mapping of the operators of the logical plan to physical hosts according to the optimization goal of the operator placement. The result is a physical plan that is finally deployed to execute the reasoning process in an overlay network of operators. During the execution of the
operators, the optimal mapping to physical hosts is adapted to changes of network conditions.

To show the feasibility of our architecture, we have implemented a first prototype of the proposed system. In particular an editor tool which enables the specification of the detectable situations has been developed and furthermore the distributed reasoning component has been realized, including operator placement mechanisms for the optimized execution of the distributed reasoning process.

The rest of the paper is structured as follows: In Section 2 we discuss the related work. In Section 3 we describe the context model. In Section 4 we give an overview of the proposed architecture. Finally in Section 5 we show the implementation details of a first prototype of the proposed architecture and in the last section we conclude our work and we set the challenges for our future work.

II. RELATED WORK

Many existing systems use centralized architectures (e.g. MS Easy Living [1] or Semantic Space [2]) to support small-scale scenarios. For instance, Chen et al. [6] proposed the CoBrA infrastructure where a centralized server called Context Broker is responsible for storing and reasoning about context information. This centralized approach cannot be extended to large-scale scenarios as it confronts scalability problems due to single point of failure or single processing bottleneck.

To overcome these problems, Ranganathan et al. [7] developed a middleware infrastructure which is based on distributed context servers called context synthesizers. The context synthesizers are spread in the network and support different reasoning methods. This approach distributes the computational load among multiple context servers. However it does not allow for the distributed execution of a reasoning task on multiple servers, and as a consequence it leads to poor network and system performance.

Przybilski et al. [8] present a first approach for distributing the reasoning process using P2P networks. In this model devices perform simple context reasoning and send their results to a remote server for more advanced context reasoning such as classification that requires more powerful computational capabilities. Although this work enables distributed context reasoning, it lacks theoretical concepts about how the reasoning task is distributed in the network and furthermore it does not include an implementation of the proposed system.

Gu et al. [5] developed a protocol for exchanging messages about context information which enables the performance of reasoning in a distributed fashion. Their system model is based on an overlay network where the peers are grouped in semantic clusters according to the type of the queries that they can answer. This approach is limited to first-order logic and their evaluation is based only to a small set of nodes (eight peers). In our model we cope with uncertain data that need more sophisticated reasoning methods and furthermore we assume large-scale overlay networks, where the distribution of the reasoning task should be done automatically according to certain optimization criteria.

Another approach [4] uses distributed Bayesian Networks and proposes a placement algorithm that clusters the nodes of a Bayesian network to minimize the communication overhead. Although this approach is closer to our work, it still only provides a solution to a method specific problem. Our goal is to create a generic formalization, where different reasoning algorithms can be distributed by different placement algorithms.

Finally relevant to our work are also distributed event processing systems such as [9], [10]. These systems use operator placement algorithms [11]–[14] to optimize the distribution of the stream processing tasks. Although these systems provide the foundation for the distribution of stream processing, they support only logical and relational operators, without allowing the use of powerful reasoning algorithms to aggregate context data. Our formalization use notions such as the operator graph, that have been developed in the field of distributed stream processing, but it combines this model with the more sophisticated reasoning algorithms used for context reasoning. The result is a system, which uses the expressiveness of the reasoning algorithms and the scalability of the distributed stream processing systems.

III. CONTEXT MODEL

In [15] context is defined as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. Based on this general definition of context, we are going to describe the modeling of observable context and situations in our system, before we will introduce our system in the next section.

A. Observable Context

Observable context is context information that can be directly observed, for instance sensor data, without further interpretation. To manage and efficiently store observable context, a powerful context model is necessary. Our work relies on the context model of the Nexus Platform [16]. In the Nexus environment context information is a part of the Augmented World, a context model that describes the world as physical and virtual objects with certain attributes and inter-object relations. Physical objects can be either static such as buildings, roads, and rooms or mobile such as people, cars, and trains. Examples for virtual objects could be virtual post-its or virtual advertising columns. The objects belong to classes that are structured in a hierarchical class schema, i.e. a car is a vehicle, which in turn is a mobile object and a Nexus object.

In the Nexus Model each context is mainly associated with a position or position in time (history). For instance virtual objects can be associated with a location in the real world, where they are relevant. All the objects in the Augmented World are modelled in Augmented World Modelling Language (AWML). Furthermore a special query language (AWQL) supporting spatial and spatial-temporal queries is used to retrieve context information form the Nexus Model.
B. Situation

A situation is composed of multiple forms of elementary context and describes the combination of circumstances at a given moment, a state of affairs. Here we use a situation-centric approach, where each situation that can be detected by the system, is predefined by experts and stored as preknowledge of the system. For each situation one or more predefined situation recognition patterns, called situation templates that describe the relations between the various pieces of context, might exist. Situation templates are graphs consisting of nodes providing observable context and operator nodes. Operator nodes are method specific and describe the processing of the input data to derive high level context. In Fig. 1 we see an example of a situation template describing the situation “Traffic Jam”. For the detection of this situation, we assume three kinds of observable context: sensor data about the number of cars in this part of the road, the average speed of the cars, and an internet text sensor which scans the WWW space to find context information related to the location of the situation. The unary operators connected with the external sources act as filters, which allow only the data within a range to pass to the next operator. Then Bayesian operators are applied to compute the probability of the (sub-)situations as described in [17]. In order to calculate the probability of a (sub-)situation, we need to know the values of the so called Contribution Probability Tables (CPT). CPTs are not predefined, but situation template might include some initial values, as shown in Fig. 1, that later will be changed by a learning process.

A situation is modelled in the Augmented World as an object which contains the state of the situation and various attributes that describe its status. The situation object includes attributes describing the quality of the detected situation. As proposed in the literature [15], [18], [19], we use two different values to define the quality of the detected situation: the confidence and the probability of the situation. On the one hand, the confidence is a measure for the correctness of the situation detection and it depends on the accuracy and the number of involved data sources. On the other hand, the probability of the situation represents the probability that this situation is taking place. The probability is computed according to the reasoning algorithm and is independent of the accuracy of the context sources. For instance, by considering only one source, the result might involve a high probability that a situation has occurred, but this definite statement might be very uncertain if the accuracy of this single source is low. Moreover the situation objects belong to classes that form a situation lattice, where relations between different situations can be expressed. Relations between situations are significant to exploit reuse of partial results e.g. “in a meeting” is a sub-situation of “in a conference meeting”, since the former is used to infer the latter. Furthermore by this approach we avoid inconsistencies between conflicting situation detections, e.g. a person can either be in a meeting or in a lecture. Further details on optimizing the situation recognition process by reusing partial results or detecting inconsistencies between situations are beyond the scope of this paper.

IV. System Architecture

Based on the situation model introduced in the previous section, we next present our system for distributed context reasoning. Fig. 2 shows the components and the interfaces of the distributed reasoning system, which belong to three different layers: World Model Layer, Context Reasoning Layer and Application Layer. In the basic layer the World Model provides the observable context to the situation detection components, which constitute the second layer that processes the observable context to derive high level context in an efficient way. The context aware application lies on the top layer, representing the user that interacts with the system either to query for situations of certain objects or locations, or subscribing for events on detected situations. In addition to these basic parts, the auxiliary services support the core components by providing additional information to situation detection components.

The system operates in two distinct phases: the Initialization Phase and the Execution Phase. During the initialization, the system creates a logical plan, which describes a reasoning task by an operator graph with pinned context sources and sinks. Then the logical plan is transformed to a physical plan, where the free operators are placed to physical nodes such that an optimization goal is achieved. The physical plan is then deployed on the physical network and the system enters the execution phase. During the execution phase the reasoning task is executed in a distributed way on the physical network and simultaneously the system adapts the physical plan to the current network condition.

Next we describe in detail the core components of the architecture:

Logical Planner. The Logical Planner receives the user specifications and it retrieves the corresponding situation template from the Situation Template Repository, which stores all the available situation templates through a distributed lookup service. The user specifications include the definition of the detectable situation as well QoC (Quality of Context) and QoS (Quality of Service) requirements of the user. As
discussed earlier, in our model QoC is expressed through the confidence and the probability of a situation. The user can set his requirements, regarding these two quality aspects and in addition set restrictions about QoS parameters such as maximum response time.

After the retrieval of the situation template, the Logical Planner contacts the Context Broker to discover the context sources needed to perform the reasoning task. For instance in the traffic jam scenario, it might ask for all camera sensors at a certain road or the context servers providing information about the average speed of cars on this road. In general the Context Broker can be realized as a distributed lookup service for context sources, where each source is described by the kind of data it provides, the quality of the provided data and the spatial area covered by the data.

In case there is more than one situation template, the Logical Planner selects one such that the quality requirements defined by the user are fulfilled. Finally the Logical Planner encapsulates each partial reasoning task in an operator, as specified in the situation template, and pins the sources and the application to their corresponding physical hosts in the network. The result of this procedure is a logical plan, which contains pinned (sources, application) and unpinned operators. This logical plan acts as an interface between the Logical Planner and the initial placement component.

Initial Operator Placement. The Initial Operator Placement assigns the unpinned operators of the logical plan to physical hosts according to defined optimization criteria. To achieve this goal, it executes an operator placement algorithm in a centralized way. In particular, the initial placement component first contacts the Resource Model to get the information about the available resources in the physical network such as latency, available bandwidth, or load. The Resource Model is dependent on the placement algorithm and can be realized as a distributed lookup service. After retrieving information about the current network condition, the initial placement should find a mapping of the unpinned operators to physical hosts which optimizes for a certain criterion. Usual criteria for placement optimizations are network usage, latency, and load [11]–[14].

Currently we use a placement algorithm, proposed in [12] that optimizes for network usage, which intuitively reduces the induced by the reasoning task network load. The output of the placement algorithm is a physical plan, which extends the logical plan with the additional information of the physical mapping of the operators. Then the physical plan is finally deployed on the physical network and the system enters the execution phase.

Distributed Operator Execution. After the deployment of the physical plan, it starts the distributed execution of the operators, which realizes the reasoning task in a distributed way. If the user has subscribed for certain situations, this task is executed permanently and the user is notified of new situations when they are detected. In particular the distributed operator execution receives the context data from the selected sources of the World Model, performs the reasoning task and then notifies the application. Furthermore it also writes the result of the context reasoning back to the World Model. As we have already mentioned, the situation is a part of the World Model and therefore its current status is to be updated. This approach also allows for the storage of historic situations.

Distributed Operator Placement. Since the network conditions might change during the Execution Phase, the initial placement might not fulfill at some point in time its optimization goal anymore. The distributed operator placement service is responsible for the adaptation of the operator placement to the current network conditions. Here the operator placement is done in a distributed way and it modifies, if necessary, a part of the physical plan. In other words, when the distributed placement algorithm finds a better placement for an operator, it initiates the migration of this operator to another physical host by modifying this part of the physical plan. Then the execution environment is responsible for the deployment of the new physical plan. This process is an event-driven process, which is triggered by changes of the network conditions. Most of the existing placement algorithms provide distributed placement strategies that adapt the operator placement during the execution of the operator graph based on local information.
For instance in [12] we proposed a distributed version of our placement algorithm optimizing for network usage.

**Feedback Adaptation.** The user can send feedback to the system about the occurrence of the detected situation in the real world (e.g. false positives/negatives). The user feedback is used by the Feedback Adaptation to improve the quality of the situation detection. In particular the feedback adaptation component is responsible for the re-configuration of the operators during the distributed operator execution. The operator configuration is dependent on the reasoning algorithm. For instance, in case of the Bayesian Networks, the algorithm proposed in [20] can be used to calculate the new values of the CPTs. Moreover apart from the adaptation of the template will be chosen by the Logical Planner according to the estimated confidence of each template. For our prototype, we always choose the template providing the user with the highest quality of situation detection. In general we assume that the individual confidence of each source and operator is measurable or computable. For instance, the quality of the output of a sensor feeding a context source can be determined through the accuracy specified in the sensor data sheet. The confidence of the (sub-)situations detected by operators is then specified by using statistical methods as described in [20].

C. Operator Placement

Here we describe the placement component for both the initial and the runtime placement. To find a mapping of the unpinned operators to physical hosts, we have implemented a placement algorithm optimizing the network usage [12]. The resource model used by this algorithm is the so called Latency Space [14], which models latencies between physical hosts in the network as Euclidean distances in a virtual coordinate space. For our prototype, the coordinates of the physical nodes in the latency space were found using a prototype implementation of the Vivaldi algorithm [23], [24] that achieves to map the physical nodes in the latency space with an average error of 15 ms w.r.t. to the measured delays. The placement algorithm tries to find node coordinates in the latency space such that the bandwidth-delay product of inter-operator data streams is minimized to optimize the induced network load. In detail, it gets the coordinates of the pinned operators (sources and application) and calculates the optimal coordinate in the latency space. Then it assigns the operator to the physical node, which is closest to this coordinate. For the initial placement, the algorithm is executed at the handler of the user request in a centralized way and it outputs the corresponding physical plan. During the operator execution, the operator placement is triggered by an event driven model due to changes of the data rates of the operators or changing latencies between physical hosts. Whenever an operator receives such an event, it calculates its new optimal coordinate in the latency space and migrates to another physical node, if this node is closer to the optimal coordinate.

D. Context Reasoning

Context reasoning includes methods for inference of situations and feedback adaptation to improve the quality of situation detection. For the realization of the distributed context reasoning, we implemented the distributed message passing algorithm of Pearl [17], which enables the distributed belief propagation in Bayesian Networks and exploits the polytree-structure of the situation templates. Moreover during the distributed execution of this reasoning algorithm, an adaptive algorithm for the correction of the CPTs is used to exploit the user feedback. In particular, we implemented the feedback algorithm described in [20], which combines methods from the field of Bayesian Networks and Machine Learning. At a first step the algorithm exploits historical data by using statistical methods to calculate the confidence of each operator. Then,
depending on the confidence of the operators the CPTs are adapted according to the current user feedback.

E. NexusDS

After the physical plan is deployed it is executed using the NexusDS execution environment service executed on a distributed set of peers. NexusDS [22] is a flexible and extensible middleware for distributed stream processing building on top of the Nexus system [25]. NexusDS was designed with a wide set of devices in mind, ranging from small mobile devices to powerful workstations in order to cope with the heterogeneous nature of distributed systems. Beyond this, NexusDS enables custom domain specific extensions in terms of services and operators, each having special requirements to the execution environment. Therefore, the NexusDS operator and service model is meta-data driven and allows reflecting highly domain specific needs. During the execution of the physical plan, each peer executes two different services, the operator execution, which realizes the partial reasoning task and the distributed placement algorithm, which adapts the placement of operators to current network conditions. Moreover in case of severe problems, such as a peer failure, NexusDS initiates an operator migration transferring the execution of operators to another execution environment instance.

VI. CONCLUSION AND FUTURE WORK

In this paper we have addressed the problem of designing a distributed system for the detection of situations in highly dynamic environments. We have introduced an architecture that enables the efficient distributed processing of context data in large-scale overlay networks. In detail we have used the operator abstraction to describe a reasoning task and we have introduced the notion of the logical plan as an interface between the reasoning and the placement algorithm. Furthermore we have presented a first prototype of our system, implementing all the vital components of the proposed architecture. As future steps, we want to enhance our distributed reasoning system by considering further aspects such as reliability and security that are not supported currently by our system.

REFERENCES
[24] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a powerful workstations in order to cope with the heterogeneous nature of distributed systems. Beyond this, NexusDS enables custom domain specific extensions in terms of services and operators, each having special requirements to the execution environment. Therefore, the NexusDS operator and service model is meta-data driven and allows reflecting highly domain specific needs. During the execution of the physical plan, each peer executes two different services, the operator execution, which realizes the partial reasoning task and the distributed placement algorithm, which adapts the placement of operators to current network conditions. Moreover in case of severe problems, such as a peer failure, NexusDS initiates an operator migration transferring the execution of operators to another execution environment instance.

VI. CONCLUSION AND FUTURE WORK

In this paper we have addressed the problem of designing a distributed system for the detection of situations in highly dynamic environments. We have introduced an architecture that enables the efficient distributed processing of context data in large-scale overlay networks. In detail we have used the operator abstraction to describe a reasoning task and we have introduced the notion of the logical plan as an interface between the reasoning and the placement algorithm. Furthermore we have presented a first prototype of our system, implementing all the vital components of the proposed architecture. As future steps, we want to enhance our distributed reasoning system by considering further aspects such as reliability and security that are not supported currently by our system.

REFERENCES