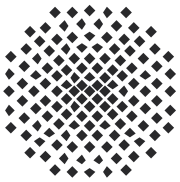

Universität Stuttgart

Sonderforschungsbereich 627

Umgebungsmodelle für
mobile kontextbezogene Systeme

SFB 627 Bericht Nr. 2010/01

Aggregation of User Context in Context-based Communcation

Datum: 11. Februar 2010

Autor(en): Lars Geiger
Frank Dürr
Kurt Rothermel

CR Klassifikation: C.2.1, C.2.2, C.2.4, C.2.6, H.3.3

Center of Excellence 627
Spatial World Models for
Mobile Context-Aware Applications

Sprecher des SFB:
Prof. Dr. Kurt Rothermel
Institut für Parallele und Verteilte Systeme
Universitätsstraße 38
70569 Stuttgart
Deutschland

NEXUS

www.nexus.uni-stuttgart.de

A context-based communication system enables the indirect addressing and routing of messages according to the users' contexts. However, for a targeted forwarding of messages towards users, the routers require a certain amount of state information. Global knowledge, i.e. each router knowing about every user, is not scalable, though, because of the necessary update messages to keep this information up-to-date.

To address this challenge, a router can aggregate similar contexts and only provide such an aggregated view to neighboring routers. However, due to different requirements and semantics, existing approaches from publish/subscribe or data clustering are not applicable in a context-based communication system. We therefore present an approach to aggregate two similar contexts, based on a similarity measure for user contexts. The algorithm can be adjusted according to the observed messages and user contexts in the system by specifying a similarity threshold to determine when contexts are aggregated.

We also present a qualitative analysis of the behavior of the system under different load scenarios and their respective appropriate similarity thresholds.

Contents

1 Introduction

Context-aware communication enables clients to disseminate messages to receivers whose context matches a given set of constraints or filters. Possible applications for such a technique include concert information for people interested in a certain musical style or invitations to study groups for students attending the same university class (cf. [GDR09]).

In a distributed system of context-aware routers, messages need to be either broadcast to reach all matching receivers or the routers need to maintain routing tables of available users and where in the network to reach them. However, it is obviously not feasible to maintain complete knowledge of all users on every router: that way, every change potentially needs to be replicated to every router in the network, i.e., every change results in a broadcast of an update message. The amount of such update messages would quickly overwhelm a system with anything but a trivial number of users. Instead, we propose that routers maintain an aggregated view of available contexts. This reduction of information enables us to only propagate certain changes to the routers, while a large part of changes is hidden by the aggregation. For example, if a router has the aggregated information that it can reach users with an *age* between 15 and 23 over a certain link, it does not affect this router when a user with the age 19 disconnects from the system.

Such an aggregation allows us to exploit a locality principle in our design: Users at a location usually share certain characteristics, e.g., many users on a university campus are of type student and within a certain age range. Also, since our routing nodes are connected according to their location, this locality principle applies to the router network as well.

By aggregating these similar contexts and only propagating an aggregated view to routers, this improves the scalability of our approach. However, while an aggregation can lower the number of update messages, it also brings a loss of information: in the example above, there might not be any contexts with an *age* between 20 and 22. A message addressed to an *age* 21 would still have to be forwarded in this direction until more detailed knowledge is available, with which a router can then prune the dissemination tree. Such messages that the system forwards without a matching receiver are called "false positives". While the aggregated views save update messages, they introduce a different kind of load into the system with these false positives. Thus, for the aggregation algorithms, we need to find a good balance between reducing the information and thus saving update messages and reducing it too much, effectively resulting in broadcast messages because the routing information is too coarse for the routers to prune the broadcast tree.

1.1 Related Work

Reducing client information is closely related to the concepts "cover" and "merge" in publish/subscribe systems. These techniques are used in systems such as Siena [CRW00] or REBECA [Müh01] to lower the communication load and state

information and thus improve the scalability of the system. However, as we have discussed in [GDR09], the semantic differences between content-based publish/subscribe and ContextCast makes it difficult to impossible to simply transfer the approaches. First, "cover" is intended for a number of subscriptions containing predicates on *ranges*, where one includes the others. The user contexts in ContextCast, however, are *points* in a contexts space, defined by the attributes and their values that make up a context. That means that any two contexts may either be identical or differ in one or more attributes, with no overlap between them. Second, a technique similar to "merge" needs to take the type of data, i.e., the different attributes of user contexts, into account. Especially a *type*-attribute based on a tree hierarchy needs to be handled differently than a quantitative attribute such as *age*. The context aggregation we present is a generalization of the concepts cover and merge.

The notion of aggregating similar contexts is also closely related to the clustering of data items, with its concepts of distance or similarity and clusters (cf. [XWI05] for an overview). Clustering algorithms that have the goal of finding a predefined number k of clusters, such as the k-Means algorithm [Mac67], are not useful in our system: it is impossible to compute the optimum number k of clusters at any given time in the system, as this depends on the messages as well as the contexts they address. And finding the best value for k by choosing a value and re-running the algorithm is computationally too expensive.

Another limitation of a number of popular clustering algorithms (e.g., the k-Means algorithm) is the need for random access to all the objects they are clustering. This means in particular that all the objects must be known at the start of the algorithm. However, in ContextCast, users can join and leave the system at any time, which requires an algorithm that can continuously add contexts to and remove them from an aggregation. The continuous addition of new elements is possible with algorithms such as BIRCH [ZRL96] or specialized stream clustering algorithms. Neither can, however, remove old elements from the data. Thus, a re-evaluation of the clustering after a context becomes invalid is not possible. The CluStream [AHWY03] algorithm employs a hierarchical micro-clustering to store a history of data items or rather micro-cluster. The authors exploit a *subtractive* property of their micro clusters to generate a clustering for an arbitrary time window from these micro-cluster. However, such a subtractive property does not hold for the user contexts in our system. Our design thus incorporates the flexibility to add and remove contexts to and from the aggregation at any time.

Furthermore, clustering algorithms are usually defined on objects in $\mathbb{R}^n$. For these, it is quite simple to calculate the distance between two objects. The contexts in our system, however, contain attribute types other than simple numerical ones, such as a structured variable based on a type hierarchy. Clustering such objects is a challenging task, as the notion of distance is not obviously defined. Gowda et al. [GD92] have researched clustering of what they call "symbolic objects", which are very similar to the contexts in our system. We use their results as a basis to derive a similarity measure for our contexts in Section 4. This allows us to develop an aggregation approach, which does not require an a priori number of clusters, can handle the

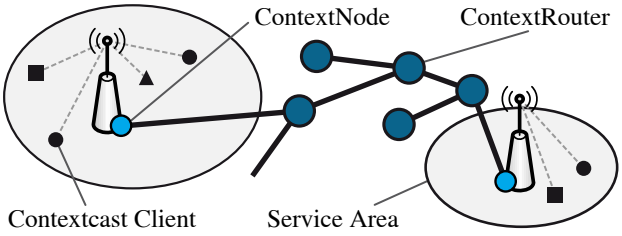

Figure 1: The CONTEXTCAST system

heterogeneous attributes in our user contexts as well as the continuous addition and removal of contexts.

2 Contextcast

CONTEXTCAST's main function is the dissemination of messages to users matching a given addressed context. To achieve this, in [GDR09] we have defined the semantics of messages, user contexts and the matching between them. The next two Sections gives a short overview of this work.

2.1 System Model

To efficiently forward messages from senders to all receivers with a matching context, we use a distributed approach with context-aware routers. Since history shows that changes to the Internet infrastructure are a slow process [CRSZ02], it seems unlikely that context-based addressing will be supported by the Internet routing infrastructure on the network layer in the near future. We therefore imagine an overlay network for this purpose, formed by infrastructure nodes.

The context-aware routers (or ContextRouters) are connected by virtual links in the overlay network. These links between routers are bidirectional and form an undirected, acyclic graph (for an arbitrary underlay, the acyclic property can always be ensured by using a routing algorithm to compute a spanning tree in the overlay). The connections between the routers follow a locality principle, i.e. connections between close routers are more likely than between ones that are far apart. This design is used to replicate an existing locality principle, e.g. a clustering of students on a campus.

In addition to the routing functionality, some of the routers are also access nodes for clients using the system. When the distinction between routers and routers with added access functionality is significant, we call the access nodes ContextNodes.

The ContextNodes maintain information about connected users and their contexts in their respective service area. This information is then propagated into the network where the ContextRouters can build routing tables with the contexts that can be reached via which one of its neighbors. However, having the complete knowledge of all users on each router seriously impairs the scalability of the system. Therefore, in

c:	WGS84: location	=	48.12N, 9.10E
	hierarchy: class	=	"pedestrian"
	enum: gender	=	"female"
	int: age	=	29
			⋮

Figure 2: Example of a user context

m:	WGS84: location	$\in$	48.0N–48.4N, 9.0E–9.2E
	enum: gender	=	"female"
	int: age	$>$	15
	int: age	$<$	35
	payload	=	[questionnaire & voucher]

Figure 3: Example of a contextcast message

Section 3 we present an approach to lower the system load by aggregating similar contexts.

2.2 Matching Messages and Contexts

A user context c in the system consists of any number of context attributes α. Let $A(c)$ denote all the attributes that make up a context c. Each attribute α is a tuple *(type, name, value)*. The location attribute α_{loc} is given as a geometric location based on WGS84, with a type of "WGS84". For the other context attributes, ContextCast currently supports the typical numerical types such as *integer* or *float*, as well as more complex types such as an enumeration *gender* or a *class* attribute based on a class hierarchy. Figure 2 shows an example of such a context c.

A (context) message m addresses clients by specifying constraints on their context attributes that need to be fulfilled to actually receive the message. These constraints (or attribute filters) ϕ are tuples *(type, name, predicate, value)*, where type_ϕ is the attribute type, name_ϕ is the attribute name, and predicate_ϕ can be any predicate that is defined on the attribute type. While notifications in pub/sub both transport information and are used by subscriptions to select notifications, the attribute filters in a contextcast message serve purely to address users. Thus, the messages have an additional payload, which is the actual message content. Figure 3 shows such a message m.

For a given attribute filter, the system can thus evaluate the constraint 'value_α predicate_ϕ value_ϕ', e.g. for the attribute *age*: $29 > 15 \rightarrow$ `true`. If a message contains an attribute filter and the attribute is not defined in a user context, the corresponding filter evaluates to `false` for that context. The conjunction of all attribute filters in a message m determines whether it *matches* a user's context c and thus needs to be delivered.

3 User Context Aggregation

In this Section, we present the aggregation of user contexts and show an algorithm to aggregate a pair of contexts. We also discuss the effects an aggregation has on a context-based communication system.

3.1 Aggregation of User Contexts

Without knowledge of user contexts, the system needs to broadcast messages (or geocast, if routers have knowledge about the access networks' service areas) to disseminate a message to all potential receiver. This is similar to a distributed publish/subscribe systems with brokers unaware of actual user subscriptions.

As in pub/sub systems, it is possible to avoid this broadcast in CONTEXTCAST. For this, the routers maintain routing tables of clients who can be reached via an overlay link. With this information, they can then prune the broadcast tree to only branches where matching receivers can be reached. However, the naive approach of propagating each user context to every router in the overlay poses a serious scalability issue. It would require that the routers maintain a tremendous amount of state information and every change of a client context would need to be propagated to every router. Therefore, we propose an aggregation approach to reduce this state information and lower the update rate.

Definition 1 (Aggregation of user contexts)**.** *Let C be a set of user contexts $c_1, \ldots, c_n$. An* Aggregation *C' of these contexts is a set of contexts $c'_1, \ldots, c'_k$ (where typically $k \ll n$) for which the following condition holds: Every message m that matches at least one context in C also matches at least one context in C'.*

This definition of a context aggregation abstracts from the actual method used to compute such an aggregation. All that is required is that the above mentioned condition holds for a given aggregation approach. It also ensures that every client receives at least all the messages they would have received without the aggregation of client contexts, i.e., no client with a matching context falsely misses a message because of the aggregation. Such a message that is not delivered despite a matching context is called a "false negative".

Aggregating contexts has positive and negative effects on the system: First, it reduces the state information that context routers need to store for the matching of messages and contexts. The smaller the number of aggregated contexts, k, is compared to the original number of contexts, n, the less state the system needs to maintain. This improves the scalability of the system, otherwise routers had complete knowledge from all connected clients. Second, it also reduces update messages, which are necessary to keep the user context information up-to-date on the routers. With the coarser context aggregations, some context changes are concealed within the aggregations. For example, if an aggregated context describes user contexts with an *age* attribute in the range $[15, 23]$ then it is not necessary to propagate the change of one context's *age* from 18 to 19 to the routers. This helps in reducing the communication load of

the overlay links. Third, since an aggregated context information is coarser than the original contexts, the system usually forwards some messages without a matching receiver. For example, consider the previously mentioned *age* range $[15, 23]$; this might be the result of an aggregation of three user contexts with an *age* of 15, 18, and 23. If now a message is addressed to people at the *age* of 21, this aggregation would cause the message to be forwarded, even though there is no single matching context and thus receiver for it. These messages without a matching receiver are called "false positives". While the first two effects benefit the system, false positive messages actually place a higher load on the overlay links because messages are forwarded without a receiver.

Based on Definition 1, we propose Algorithm 1 to aggregate two contexts into one: It aggregates all attributes that occur only in either c_1 or c_2 directly into c_{aggreg}. For

Algorithm 1 Pairwise Context Aggregation

Require: Two user contexts c_1 and c_2 (singleton contexts or already aggregated ones).
Ensure: An aggregated context c_{aggreg}.

```
c_aggreg ← ∅
for all α_{i,1} ∈ A(c_1) do
  if ∃α_{j,2} ∈ A(c_2) : name(α_{j,2}) = name(α_{i,1}) then
    α_merged ← α_{i,1}
    value(α_merged) ← value(α_{i,1}) ∪ value(α_{j,2})
    c_aggreg ← c_aggreg ∪ α_merged
  else
    c_aggreg ← c_aggreg ∪ α_{i,1}
  end if
end for
for all α_{j,2} ∈ A(c_2) do
  if α_{j,2} ∉ c_aggreg then
    c_aggreg ← c_aggreg ∪ α_{j,2}
  end if
end for
```

all attributes that occur in both contexts, it merges the values and then aggregates the resulting new attribute into c_{aggreg}. This merge of attribute values needs to be defined for the different attribute types in our system. For example, for WGS84 coordinates, a merge of two values is the bounding rectangle of both positions. This definition can also be applied when one or both positions are shapes instead of points (when aggregating already aggregated contexts). Or for ordered, numerical types, the merge of values can be either the set resulting from the union of all values or it can be a closed interval from the smallest to the largest value. Again, this definition is sound for single values as well as for ones that are already sets or intervals. In addition, to ensure that the aggregation condition holds, the predicates used in attribute filters must be adapted to this definition. In detail, we have to check now whether the

addressed attribute values occur in the values of aggregated attributes, which may now contain ranges or sets instead of single values. This is also rather straight-forward, for example for a numerical type aggregated to an interval the equality predicate must check whether the interval contains the specified value. Or in the case of a set, it must compare the specified value with all elements from a set. Similarly, for predicates such as > 5, the evaluation can check if the context interval or set contains a value for which the predicate evaluates to *true*.

It can be shown that the aggregation condition holds for any context that results from the (repeated) application of Algorithm 1 on a set of contexts: First, for any pair, the resulting context contains the union of all attributes from the two original contexts. Second, each of the attributes α_i in the resulting context is either identical to one from one of the original contexts (if it occurred in only one of the contexts) or was merged from the two original ones. This merging is done in such a way that any attribute filter that matches this attribute in one of the original contexts also matches the corresponding attribute in the aggregated context.

The algorithm can be applied to both singleton contexts or already aggregated ones. Thus, by repeated execution for pairs of user contexts, it can aggregate any given set of contexts $C = \{c_1, \ldots, c_n\}$ into a single context c', for which the aggregation condition holds. □

3.2 Communication Overhead of a Context Aggregation

The communication load of a context aggregation is defined by the amount of messages in the system. This includes both context messages and maintenance messages. Content messages are addressed to users and we can distinguish between true positives, i.e., messages that have to be forwarded to one or more matching receivers, and false positives, i.e., messages forwarded without a matching receiver. Maintenance messages are used internally by the system to ensure the correct and/or optimized operation of the system, e.g., update messages to maintain the routing tables of ContextRouters.

In ContextCast, by delivering messages along a spanning tree and not sending duplicates over individual links, we can optimize the amount of true positive messages. However, there is a lower limit to the amount of these messages, since we need to deliver a message to every access node with one or more matching receivers. The communication overhead, i.e., the amount of false positive messages and update messages, though, depends on the aggregation of user contexts. In a good aggregation, both values should be small. We therefore aim for an aggregation that minimizes this communication overhead.

The overhead depends mainly on the chosen aggregation. For instance, consider the previous example: if an algorithm aggregates two completely unrelated contexts, such as a person at age 8 with one at age 80, intuitively, the aggregation causes a many false positives. It is thus worse than one where two very similar contexts have been aggregated, such as age 19 and 20. Therefore, an algorithm needs to

aggregate "similar" contexts. To achieve this goal, we need a formal way to measure the similarity of contexts.

Related to that is the granularity of an aggregation, i.e., the ratio between n and k. If we assume that an algorithm aggregates perfectly, i.e., the most similar contexts, then the overhead of the aggregation is determined by the granularity of the aggregated contexts. In the extreme case of a single context per user, the routers have the most precise information, thus the amount of false positives is zero. However, if any of the contexts change, this change needs to be propagated into the overlay network. In the other extreme of a single context for all users, the information loss is maximized, one single context representing all the users in the system. This means that the system generates a large amount of false positives because of the uncertain context information. However, many of the original contexts are concealed within the aggregation, thus it reduces the amount of update messages that are propagated into the network.

Therefore, to compute a good aggregation, we need an algorithm that finds and aggregates the most similar contexts and achieves a granularity that optimizes the combined load of update and false positive messages.

4 Continuous Context Aggregation

In this Section, we present our algorithm to continuously aggregate new and changed user contexts in the ContextCast system and thus improve overall system scalability. Before we can present our algorithm, we need to define a similarity measure between contexts. After that, we discuss the effect of attribute popularity and how to exploit it to improve our algorithm further. These two are important building blocks for our continuous aggregation algorithm, which we introduce in the last Subsection.

4.1 Context Similarity

Figure 4 shows an example for two contexts c_1 and c_2. As one can see, these contexts share attributes α_2 and α_4 — although with different values —, while α_1 and α_3 are only part of c_1 and c_2, respectively.

c_1:				c_2:			
	α_1	$=$	v_1				
	α_2	$=$	v_2		α_2	$=$	v_3
					α_3	$=$	v_4
	α_4	$=$	v_5		α_4	$=$	v_6

Figure 4: Example of two contexts c_1 and c_2

The aggregation approach we introduced in Section 3.1 aggregates this into the user context shown in Figure 5.

As one can see from this example, such an aggregation introduces two types of uncertainty into the system: First, for attributes that are only part of one context, an

$$
\begin{array}{lrcl}
c_{\text{aggreg}}\text{:} & \alpha_1 & = & v_1 \\
& \alpha_2 & = & [v_2; v_3] \\
& \alpha_3 & = & v_4 \\
& \alpha_4 & = & [v_5; v_6]
\end{array}
$$

Figure 5: Aggregation of contexts c_1 and c_2

aggregation introduces new attribute combinations into the system. In this example, a message addressed to the combination of α_1 and α_3 (and their respective values) would not have matched the precise, unaggregated contexts c_1 and c_2, where each context contains only either α_1 or α_3. It matches the aggregated context c_{aggreg}, though. Second, for attributes that appear in both contexts, it introduces an uncertainty for the values of these attributes. In the example, a message addressed to $\alpha_2 = v$ with $v_2 < v < v_3$ would not have matched either of the two contexts, however, since $v \in [v_2; v_3]$, it matches the aggregated context.

Based on this observation and the notion that aggregating "similar" contexts should minimize the uncertainty that is introduced in the aggregation, we can derive a similarity measure for our system. As the example shows, the two aspects that determine context similarity are the set of attributes forming the two contexts (or rather, the overlapping attributes) as well as the values of the overlapping attributes. Thus, we can distinguish between a structural similarity and value similarity.

4.1.1 Structural Similarity

The structural similarity of contexts determines how similar the attribute sets of two contexts are before aggregating and thus also how much information is added in an aggregated context. As we have shown in the previous section, all attributes that only occurred in one of the contexts introduce new combinations when aggregating the two. Obviously, the more attributes are *not* shared between two contexts, the more possible attribute combinations result from aggregating the contexts, which were not present before.

The structural similarity can thus be measured by the relative overlap in attributes between to contexts: on the one hand, if we have two contexts with 100 attributes each and only a single attribute occurs in both, the relative overlap and thus the structural similarity is tiny; an aggregation of these two results in a new context with 199 attributes and a huge amount of previously not existing attribute combinations. If, on the other hand, each context has only two attributes, a shared attribute between them is a big overlap; if we aggregate them, the resulting context has only three attributes. With $A(c)$ denoting the set of all attributes making up c, we can express the structural similarity $S_{\text{structural}}$ formally as:

$$
S_{\text{structural}}(c_1, c_2) = \frac{|A(c_1) \cap A(c_2)|}{|A(c_1) \cup A(c_2)|} \tag{1}
$$

4.1.2 Similarity of Single Attribute Values

As we have argued in the previous section, the relative overlap between contexts determines how much new information an aggregation of two contexts introduces into the system, i.e., previously unknown attribute combinations. For the overlapping attributes, i.e., the attributes that appear in both contexts, the similarity of the values determines how much uncertainty is added to a single attribute when the values of two different contexts for this attribute are aggregated.

In CONTEXTCAST, we distinguish between the following types of attributes (or features), each with its own similarity measure, taken from [GD92]:

- Quantitative Features
  - continuous ratio values, e.g., size or velocity.
  - discrete absolute values, e.g., objects.
  - interval values, e.g., duration.
- Qualitative Features
  - nominal (unordered), e.g., color or gender.
  - ordinal (ordered), e.g., designation.
- Structured Variables
  - tree-ordered sets such as a type hierarchy.

The authors in [GD92] and subsequently [GR95] describe how to compute the similarity for attributes of these types, with the exception of structured variables, which they mention as one possible type but never detail. From this information, they then compute the similarity between two symbolic objects by adding the similarity values of all attribute-wise similarities. We closely follow the approach in [GR95] for attribute similarity. However, there are two notable differences: first, as we have shown in Section 4.1.1, the attribute sets that make up two contexts are important for the structural similarity of these contexts in CONTEXTCAST. In the related work, the authors mention that the composite objects need not be defined on the same set of attributes but never detail how that affects the computation of similarity. We show in Section 4.2 how to use both the structural and value similarity of two contexts to compute their overall similarity. Second, they do not describe a similarity measure for structured variables, which are an important part in our system.

According to [GR95], for each attribute α_k in both contexts D and E, we can compute the value similarity as the sum of three components:

1. $S_p(D_k, E_k)$: Similarity due to position p
2. $S_s(D_k, E_k)$: Similarity due to span s
3. $S_c(D_k, E_k)$: Similarity due to content c

The similarity due to *position* is used to describe the relative position of two values on a real line. The component due to *span* measures the relative size of two attribute values, without considering common parts. The similarity due to *content* indicates the common parts between two values.

The components for *position*, *span*, and *content* are all in $[0, 1]$. The overall value similarity of the k-th attribute is thus defined as the sum of these three components, divided by the number of components to normalize the result to $[0, 1]$:

$$S(D_k, E_k) = \frac{S_p(D_k, E_k) + S_s(D_k, E_k) + S_c(D_k, E_k)}{3}. \tag{2}$$

Quantitative Attributes. According to [GD92, GR95], the similarity of quantitative interval types can be computed from the similarity due to *position* and *span*: The similarity due to *position* is defined as

$$S_p(D_k, E_k) = 1 - \frac{|d_l - e_l|}{u_k},$$

with d_l, e_l the lower bounds of D_k, E_k, respectively, and u_k the maximum interval of the attribute k. Similarity due to *span* is defined as

$$S_s(D_k, E_k) = \frac{l_d + l_e}{2 \cdot l_s},$$

with l_d, l_e the length of the intervals D_k, E_k, respectively, and l_s the span length of intervals D_k, E_k, i.e., $|max(d_u - e_u) - min(d_l - e_l)|$, where d_u, e_u are the upper bounds of D_k, E_k, respectively.

The resulting normalized similarity for quantitative interval types is therefore

$$S(D_k, E_k) = \frac{S_p(D_k, E_k) + S_s(D_k, E_k)}{2}. \tag{3}$$

The other quantitative types of attributes, ratios and absolute values, are special cases of the quantitative interval type, with $d_l = d_u$, $e_l = e_u$, $l_d = l_e = inters = 0$.

Qualitative Attributes. As shown in [GD92, GR95], the similarity of qualitative attributes can be measured as the similarity due to *span* and *content*. Let l_d be the length or number of elements in D_k, l_b accordingly for B_k, inters = number of elements common to D_k and E_k, and l_s = span length of D_k and E_k combined $= l_d + l_e -$ inters. The similarity due to span is then defined as:

$$S_s(D_k, E_k) = \frac{l_d + l_e}{2 \cdot l_s}.$$

The similarity due to content is defined as:

$$S_c(D_k, E_k) = \frac{\text{inters}}{l_s}.$$

For qualitative attribute types, this results in a total value similarity of

$$S(D_k, E_k) = \frac{S_s(D_k, E_k) + S_c(D_k, E_k)}{2}. \tag{4}$$

Structured Attributes. The value similarity of a variable that is based on a tree-structure, such as a type hierarchy, is computed as the similarity due to position using the distance in the tree and the overall size of the tree:

Let n_1, n_2 be two nodes that correspond to D_k and E_k in a given tree structure and h the height of this tree. Let also $l(n)$ denote the level of a node n in this tree. With this, the similarity between n_1 and n_2 is computed as follows: Find the first common ancestor a for n_1 and n_2.

$$S(D_k, E_k) = S_p(n_1, n_2) = 1 - \frac{(l(n_1) - l(a)) + (l(n_2) - l(a))}{2 \cdot h} = 1 - \frac{l(n_1) + l(n_2) - 2l(a)}{2 \cdot h} \quad (5)$$

4.1.3 Value Similarity

Let (without loss of generality) $\{\alpha_1, ..., \alpha_k\}$ be the set of all attributes shared between two contexts c_1 and c_2. Also let $\{\alpha_{k+1}, ..., \alpha_n\}$ be the remaining attributes that occur only in one of the contexts. With the attribute-wise similarities $S(c_{1,i}, c_{2,i})$ for the i-th attribute, one could then compute the value-similarity between the two contexts, $S(c_1, c_2)$ simply as the sum of the similarity of all the overlapping attributes:

$$S_{\text{value}}(c_1, c_2) = \sum_{i=1}^{k} S(c_{1,i}, c_{2,i}) \quad (6)$$

4.2 Overall Similarity of Contexts

The overall similarity between two contexts can be described by two components: first, the structural similarity between to contexts, i.e., the relative overlap of context attributes between the two; second, the similarity due to the values of the shared attributes.

For the total similarity, it seems natural to multiply the attribute similarity with the relative overlap to modify the similarity due to attribute values by the structural similarity of two contexts:

$$S(c_1, c_2)' = S_{\text{structural}}(c_1, c_2) \cdot S_{\text{value}}(c_1, c_2) = \frac{k}{n} \sum_{i=1}^{k} S(c_{1,i}, c_{2,i}) \quad (7)$$

This matches the intuitive notion that for contexts with identical attribute sets, thus $S_{\text{structural}}(c_1, c_2) = 1$, only the value similarity is important, while for contexts with disjoint attribute sets, and thus $S_{\text{structural}}(c_1, c_2) = 0$, the overall similarity is 0, independent of the value similarity (which is also 0, since there are no overlapping attributes for which to compute the value similarity). For all cases between these extremes, the value similarity is multiplied by the structural similarity with $0 < S_{\text{structural}} < 1$.

However, the value similarity depends on the actual number of attributes occurring in both contexts. The higher the number k of shared attributes, the higher the attribute similarity will be. Therefore, we can modify Equation 7 to instead use the arithmetic mean of the value similarity:

$$S(c_1, c_2) = S_{\text{structural}}(c_1, c_2) \cdot \frac{1}{k} S_{\text{value}}(c_1, c_2)$$
$$= \frac{k}{n} \frac{\sum_{i=1}^{k} S(c_{1,i}, c_{2,i})}{k} = \frac{1}{n} \sum_{i=1}^{k} S(c_{1,i}, c_{2,i}) \quad (8)$$

4.3 Attribute Popularity

The similarity definition from the previous Section does not take the popularity of individual attributes into account. This information helps in deciding which attributes to aggregate more aggressively (the rarely used ones) and which to propagate very detailed into the network. For example, for a rarely used attribute, it may be enough to have the knowledge that the attribute occurs in the direction of a given link, with only a very broad knowledge of the values (e.g., $age = [8; 80]$. If such an imprecise information is the only information for an attribute that is used in 95% of all messages, the amount of false positives would be very high; if the attribute is used in only a few messages, the number of false positives is nonetheless low.

More formally, for each attribute α_i, the system counts the number of messages that use a constraint on α_i, m_{α_i}, during a window t_w, as well as the total number of messages, m. The relative frequency of an attribute α_i in the messages during t_w can then be used as a measure for the popularity P of α_i:

$$P_{\alpha_\text{i}} = \frac{m_{\alpha_\text{i}}}{m}$$

Intuitively, according to the previous discussion, the more popular an attribute is, the less it should be aggregated and vice versa.

This can be combined with the similarity from the previous Section to create the weighted similarity: The similarity for an attribute can be weighted by the reciprocal of the popularity, thus lowering the chance for two contexts to be aggregated if they contain a popular attribute with values that are not very similar. The formula for the weighted similarity of two contexts c_1 and c_2 therefore becomes

$$S_{\text{weigthed}}(c_1, c_2) = \frac{1}{n} \sum_{i=1}^{k} \frac{1}{P_{\alpha_i}} S(c_{1,i}, c_{2,i}) \quad (9)$$

4.4 Continuous Aggregation Algorithm

In ContextCast, user contexts are continuously added to and removed from the system. While Algorithm 1 allows us to aggregate pairs of contexts under the aggregation condition, it does not select which pairs of contexts to aggregate or

handle the continuous addition and removal of contexts. The following two algorithms provide that.

Context Addition. With the context similarity measure from the previous sections, the system can compute the similarity between pairs of contexts, either singleton contexts or aggregations. Together with Algorithm 1, we can then aggregate newly added contexts with existing ones as follows: When a router receives a newly added context c_{add} from one of its neighbors, it can compute the similarity between c_{add} and all other contexts associated with that link. If the highest similarity between c_{add} and an existing context c_{ex} then exceeds a configurable threshold similarity, s_{th}, the system can aggregate this new context with the existing one using Algorithm 1. If the resulting context c_{new} differs from c_{ex}, c_{new} must be propagated as an update for c_{ex}. In this case, we call c_{add} a "defining context" for c_{new} (this is important for the context removal) and also mark the attributes that changed or were added by the aggregation of c_{ex} and c_{add}. Algorithm 2 shows this sequence formally.

Algorithm 2 Context Addition

Require: A newly added context c_{add}, received via a link l.
Ensure: c_{add} is attached to the link l.

```
s_max ← 0
for all c_i ∈ {contexts already attached to l} do
  s_i ← s(c_add, c_i)
  if s_i > s_max then
    s_max ← s_i
    max ← i
  end if
end for
if s_max > s_th then
  c_new ← merge c_i and c_add
  if c_new differs from c_i then
    Mark c_add as defining context for c_new.
    Replace c_i on l with c_new.
    Propagate c_new as update/replacement for c_i on all links except l.
  end if
else
  Attach c_add as singleton context to l.
  Propagate c_add on all links except l.
end if
```

Context Removal. The removal algorithm for contexts (e.g., when a user disconnects rom the system) is rather similar to the addition algorithm. Basically, when a context c_{rem} is supposed to be removed from a link, a node first checks whether c_{rem} is a singleton context or part of an aggregation. A singleton context can simply be removed and its removal propagated to neighbors. If the context is part of an aggregation, it depends if it is a defining context for this aggregation. If it is not, its

removal does not affect this context. If it is a defining context, the algorithm needs to recalculate the affected attributes, and propagate this update for this context. See Algorithm 3 for a formal description of these steps.

Algorithm 3 Context Removal

Require: A context c_{rem} to remove from a link l.
Ensure: c_{rem} is removed from the link l.
  if c_{rem} is a singleton context **then**
    Remove c_{rem} from l.
  else {c_{rem} is part of an aggregation c_{aggreg}}
    if c_{rem} is defining context for c_{aggreg} **then**
      Recalculate the affected attributes of c_{aggreg}.
      Propagate an update of c_{aggreg} on all links except l.
    else
      Do nothing.
    end if
  end if

Similarity Threshold. The similarity threshold serves as a parameter for the operation of Algorithm 2. It determines how similar two contexts must be before they are aggregated and thus whether the system has a very fine or a rather coarse aggregation. By adjusting its value, either manually by a human administrator or autonomously, one can optimize the system for the observed contexts and messages.
Optimized Calculation of Context Similarity. For a newly added context, a node must calculate its similarity with all other contexts that are attached to the same link. Especially when there are already a large number of contexts attached to a link, this calculation can cause a rather large amount of load for the node. However, it is possible to simplify this calculation.

The similarity of two contexts consists of the product of the structural and the value similarity. By employing a concept similar to Bloom filters [Blo70], it is possible to calculate the structural similarity very easily. Let b_i be a bit string for a user context c_i. Every attribute in c_i can then be hashed to a position in b_i, which is then set to 1. Thus, b_i is a representation of the structure of c_i. Also, let $|b_i|$ be the number of 1 bits in b_i. Then, the structural similarity between c_{ex} and c_{add} can be expressed as

$$S_{\text{structural}}(c_{\text{ex}}, c_{\text{add}}) = \frac{|A(c_{\text{ex}}) \cap A(c_{\text{add}})|}{|A(c_{\text{ex}}) \cup A(c_{\text{add}})|} = \frac{|b_{c_{\text{ex}}} \vee b_{c_{\text{add}}}|}{|b_{c_{\text{ex}}} \wedge b_{c_{\text{add}}}|} \tag{10}$$

Binary AND and OR can be computed very fast on conventional computer systems, the hashing of attribute names to bit positions can be done once and cached.

From this structural similarity, we can now derive a set of candidate contexts as follows: for a given similarity threshold S_{th}, we exclude a context from the candidate set (and thus skip the calculation of the value similarity) if the structural similarity is smaller than S_{th}: From $S = S_{\text{structural}} \cdot S_{\text{value}} > S_{\text{th}}$, we can derive that also

$S_{\text{structual}} > S_{\text{th}}$. Thus, we only need to actually calculate the value similarity for those candidates with a high enough structural similarity.

5 Effects of the Similarity Threshold

The Similarity Threshold determines the granularity of the aggregation and can thus be used to adjust the aggregation algorithm to the observed load of messages and context updates. This adjustment can be performed either by a human administrator or autonomously, when the observed load in the system is unfavorably high for either false positives or context updates.

A high value for s_{th} (i.e., close to 1) makes the algorithm only aggregate very similar contexts. This leads to very detailed context information on the routers, which in turn can make very precise forwarding decisions. As a result, the amount of false positives in the system is low, however, many update messages need to be propagated deeply into the network. A lower value for s_{th} (i.e., closer to 0) results in a more aggressive aggregation of user contexts. On the one hand, this reduces update messages since the aggregations conceal some of the updates. On the other hand, the reduced information on the routers results in an increase of false positive messages.

The optimal value for s_{th} depends on numerous factors, especially the user contexts in the system and the addressing of messages. For now, we assume that a human administrator can set a reasonable value for s_{th} that the system operates near the optimal point of combined false positive and update load. A quantitative evaluation of reasonable threshold values under different scenarios (e.g. a system load dominated by updates vs. one dominated by messages) is currently under way.

6 Conclusion & Outlook

An aggregation of user contexts in Contextcast can be used to lower the load on the system and thus improve the overall scalability.

We have shown an approach that allows us to continuously aggregate user contexts in Contextcast, while still being able to remove old contexts from the system. It uses a similarity measure for contexts to decide when a new context has to be aggregated with an existing one. The similarity measure takes into account both the structure of user contexts, i.e. which attributes are present, and the values of the attributes, i.e. how close are two attributes in the context space.

The similarity threshold s_{th} allows us to tune the aggregation algorithm to the contexts and messages that occur in the system. We are currently investigating sensible values for s_{th} under different scenarios.

In the future, we are going to improve the presented algorithm by allowing the routers in the system to autonomously adapt the algorithm according to the amount of overhead placed on them.

Acknowledgments

The work described in this paper was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627.

References

[AHWY03] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *vldb'2003: Proceedings of the 29th international conference on Very large data bases*, pages 81–92. VLDB Endowment, 2003.

[Blo70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[CRSZ02] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *Selected Areas in Communications, IEEE Journal on*, 20(8):1456–1471, Oct 2002.

[CRW00] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Content-Based Addressing and Routing: A General Model and its Application. Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, Boulder, CO 80309, USA, jan 2000.

[GD92] K.C. Gowda and E. Diday. Symbolic Clustering Using a New Similarity Measure. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(2):368–378, Mar/Apr 1992.

[GDR09] Lars Geiger, Frank Dürr, and Kurt Rothermel. On Contextcast: A Context-Aware Communication Mechanism. In *Proceedings of the IEEE International Conference on Communications (ICC '09)*, 2009.

[GR95] K. Chidananda Gowda and T. V. Ravi. Agglomerative clustering of symbolic objects using the concepts of both similarity and dissimilarity. *Pattern Recognition Letters*, 16(6):647–652, 1995.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. LeCam and J. Neyman, editors, *Proc. of the 5th Berkeley Symp. on Mathematics Statistics and Probability*, 1967.

[Müh01] Gero Mühl. Generic Constraints for Content-Based Publish/Subscribe. In *Cooperative Information Systems*, Lecture Notes in Computer Science, pages 211–225. Springer, Berlin and Heidelberg, 2001.

[XWI05] Rui Xu and Donald Wunsch II. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, may 2005.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 103–114, New York, NY, USA, 1996. ACM.