Similarity-Based Resource Retrieval in Multi-Agent Systems by Using Locality-Sensitive Hash Functions

Malte Aschermann, Jörg P. Müller

Abstract. In this paper we address the problem of retrieving similar resources which are distributed over a multi-agent system (MAS). In distributed environments identification of resources is realized by using cryptographic hash functions like SHA-1. The issue with these functions in connection with similarity search is that they distribute their hash values uniformly over the codomain. Therefore such *ID*s cannot be used to estimate the similarity of resources, unless one enumerates the whole search space and retrieves every resource for comparison. In this paper we present a three-layer architecture and a data model to efficiently locate similar resources in linear time complexity by using *locality-sensitive hash functions*. We design the data model as an extension to distributed environments (MAS), which only need to provide at least basic resource management capabilities, such as storing and retrieving resources by their *ID*. We use a benchmark data set to compare our approach with state-of-the-art centralized heuristic approaches and show that, while these approaches provide better search accuracy, our approach can deal with decentralized data and thus, allows us to flexibly adapt to dynamic changes in the underlying MAS by distributing and updating sets of information about similarities over different agents. locality-sensitive hash functions, multi-agent systems, similarity search

1. Introduction

In many real-world application contexts, such as product development [**SHM12**] or medical diagnosis [**WSHM95**], information relevant for decision-making is distributed across systems and organizations. For instance, companies rely on efficient product data management systems (PDM) to share specifications and models with their suppliers. To date, the client-server approach is prevalent for organizing this type of systems. We found that, while it is efficient for a limited set of participants and especially if security concerns (e.g. intellectual properties) are an issue, it will not scale well for larger amounts of data distributed over larger number of partners, which need access to minor, overlapping subsets of the data and additionally need to exchange information with participants to satisfy e.g., construction-related constraints. This problem becomes even worse because suppliers need to store a rapidly growing number of product revisions of similar, but mutually compatible parts, assemblies or even whole modules. In [**SHM12**], we proposed a Product Collaboration Platform (PCP) – based upon the FreePastry [**DEG$^+$12**] overlay system – supporting decentralized product development; we showed that this approach can outperform the single-server model. A very important aspect of decentralized architectures lies in the distribution of data objects by the overlay network, thus a participant can easily retrieve all the information about a specific product specification if he knows its identifier. One aspect not considered in [**SHM12**] was the fact that in product collaboration environments the exact characteristics of needed parts, assemblies or even modules won't be necessarily known during development; the ability to locate specifications that are similar to a given specification can reduce development time. However, distributed semantic similarity-based search exceeds the scope of existing peer-to-peer (P2P) overlay systems such as Pastry, as it requires more computational intelligence and flexibility at the partners sides. Therefore, we are moving from peer-to-peer systems towards multi-agent systems (MAS), incorporating richer local computation models as well as more flexible modes of interaction. A practical scenario would be the collaborative development of a twin engine, which can be seen in Fig. 1. The blue highlighted activities display processes for locating specifications of similar and compatible parts or assemblies, the red activity references a subprocess, which, for matters of simplicity, is not shown. As shown in the activity diagram, it is – during the planning phase – reasonable to

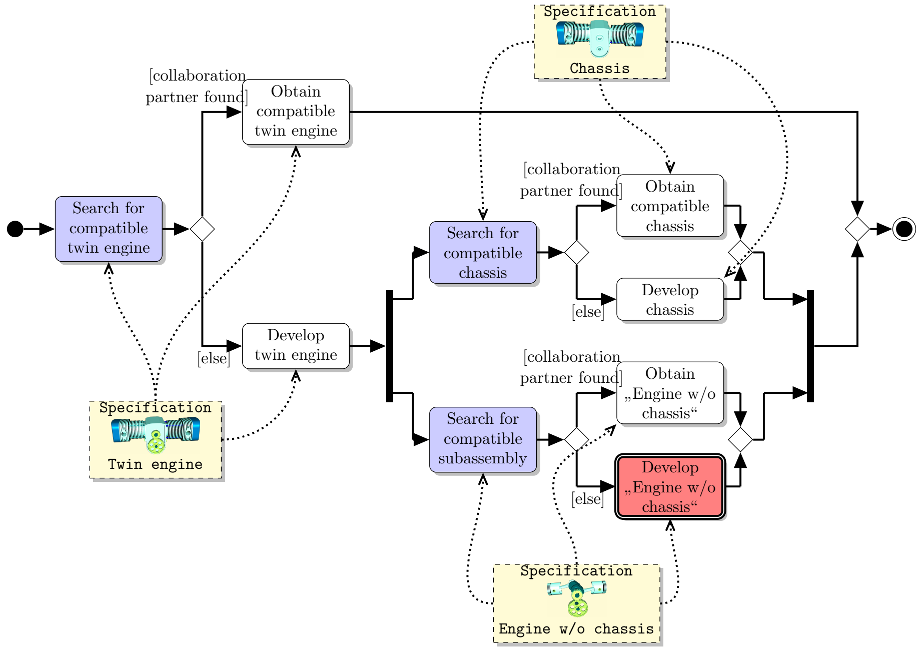
Specification
Chassis
[collaboration partner found]
Obtain compatible twin engine
Search for compatible twin engine
Develop twin engine
[else]
Specification
Twin engine
Search for compatible chassis
[collaboration partner found]
Obtain compatible chassis
Develop chassis
[else]
Search for compatible subassembly
[collaboration partner found]
Obtain „Engine w/o chassis“
Develop „Engine w/o chassis“
[else]
Specification
Engine w/o chassis

FIGURE 1. Collaborative development of a twin engine (parts/assemblies from [**Sie12**])

look for parts or assemblies (e.g. the chassis), which have already been developed beforehand and are similar enough to be compatible with an initial specification. This allows us to restrict the search space to similar specifications, which are, however, distributed across participants in the collaboration network. By incorporating P2P lookup techniques like distributed hash tables (DHT) into multi-agent systems (MAS), we can accomplish this efficiently in logarithmic time (e.g. binary search in trees, finger tables). However, this only works well if the hash value of the resource is known or can be trivially computed by using a previously known copy of the resource. Thus, in particular, existing methods cannot cope well with the requirements of similarity search. In this paper we design a novel general-purpose *local sensitive data model* and present an efficient distributed algorithm for locating resources similar to given *prototypes*. We compare the lookup performance of our approach with state-of-the-art centralized heuristic classification approaches, such as k-means or neural gas. In the following we will assume that an underlying decentralized infrastructure (MAS) already exists, and therefore can trivially locate resources if their exact identifier (i.e. hash-value) is known to the searching party. Thus we will elaborate how product collaboration between different participants, i.e. agents, can be optimized by introducing specialized types of agents, which dynamically keep track of similar data (i.e. resource specifications), therefore minimize the search-space and allowing to locate similar data in linear time.

This paper is structured as follows: In Section 2, we discuss related work. Section 3 provides a short introduction to similarity search by using locality sensitive hash functions. In Section 4 we present the locality sensitive data model; a qualitative and experimental evaluation is given in Section 5. Section 6 concludes with a discussion of results and outlook to future work.

2. Related Work

There is a rich body of work on grouping resources and optimizing data management with respect to availability and minimal lookup-time for certain data clusterings (e.g. by topics or similarity in general). In this section, we discuss three approaches for large-scale management of resources in distributed environments: OceanStore, Kademlia and Squid.

OceanStore The OceanStore architecture [**KBC$^+$00**] is a resource management system for securely storing sensitive information in untrustworthy environments and to guarantee high availability even in very large and highly distributed networks. This is accomplished by using a technique called *promiscuous caching*, to keep as many data copies as possible distributed in the whole network at a time. Therefore OceanStore is very well suited to be used in networks where participants are scattered over large distances and resources belonging to certain users need to be made available very fast depending on the user's current location. Though this concept comes with a high reliability and availability of data, the identification of resources is done by using cryptographic hash functions like SHA-1 [**KBC$^+$00**, p. 3]. Given the fact that it is infeasible to conclude the content of data just by knowing a hash value, one would need to completely enumerate the search space to find similar resources. Therefore this approach is not suitable for a similarity-sensitive localization.

Kademlia The Kademlia protocol as a method for content sensitive resource management has been introduced by Maymounkov and Mazières in [**MM02**]. The authors are using lookup information which are provided by distributed hash tables to find resources efficiently. The content sensitive localization is achieved by using an xor-metric to compare distances between the *ID*s of participants and hash values of resources. Those *ID*s, whose hash values have a minimal distance to a participant, are assigned to it accordingly. Therefore requests for data objects with similar *ID*s can be answered quite efficiently. The problem with this approach also lies in the generation of *ID*s, which are – equally to OceanStore – SHA-1-based. For this reason Kademlia is not suited for grouping resources by mutual similarity of their content and does not provide an efficient way to look for similar data.

Squid A third quite interesting approach for similarity-based data management is an extension of the P2P overlay network *Chord*. In their work Schmidt and Parashar [**SP03**] address the issue that due to the homogenous distribution of hash-*ID*s, the locality of similar resources can not be accounted for. The authors propose an extension to Chord to sustain the locality of resources [**SP03**, p. 228]. They use multi-dimensional key-spaces and thus allow lexicographic searches to neighboring resources by using *Hilbert space-filling curves*. Squid is a promising approach to locate similar data-objects in a distributed environment. However, Shu et al. point out, that *"Squid [...] partition[s] the space statically"* and that *"In Squid [...], the partitioning level needs to be decided beforehand."* [**SOlTZ05**, p. 2]. In the context of this paper we will present a method in which no static partitioning occurs, but rather a dynamic assignment of similar resources to so called *buckets* (see Section 4) is made. We will show that this approach is far better suited for the dynamic nature of distributed environments with autonomous participants.

3. Similarity-Based Localisation

Identification, management and localisation of resources in distributed environments is typically done by using cryptographic hash functions (e.g. SHA-1) which leads to homogenous distributed *ID*s in the range of possible hash values. Although intended to minimize the probability of hash collisions, it makes finding similar resources by using such *ID*s as lookup information problematic. To find similar resources one could try to compute $\text{diff}(h^{-1}(\text{ID}_{x_i}), x_{\text{ref}}) \rightarrow \min!$ for each available resource *ID* and a given reference x_{ref}; doing so, however, is infeasible due to the properties of cryptographic hash functions. Alternatively, one would need to request every available resource and locally compute the differences between them and the reference object. If the difference is below a given threshold – respectively the similarity above – a candidate is found. This approach, though significant faster than computing h^{-1}, would still need quadratic time complexity $(O(n(n-1)))$ to find the similarities between each of n existing resources.

In this paper we propose the use of *locality-sensitive hash functions*[1] (LSH), as originally presented by Koga et al. [**KIW04**] to address this problems. In contrast to cryptographic hash functions, LSH provides collision maximising properties. Their idea is, that similar values of vectors $x \in \mathbb{R}^n$ map to the same hash-value $g(u_x) \in \{0,1\}^k$. To accomplish this, hash functions $g_i \in \mathcal{H}$ are applied to elements of a data-set $\mathbb{X} \subset \mathbb{R}^n$, which map vectors x to hash values. Multiple functions $(l = |\mathcal{H}|)$ are used in order to increase the likelihood of collisions and identify similar data-points. For using vectors of $\mathbb{R}^n$ to represent product specifications, resources have to be transformed into vectors, by using specific characteristics (e.g. measurements) as vector attributes.

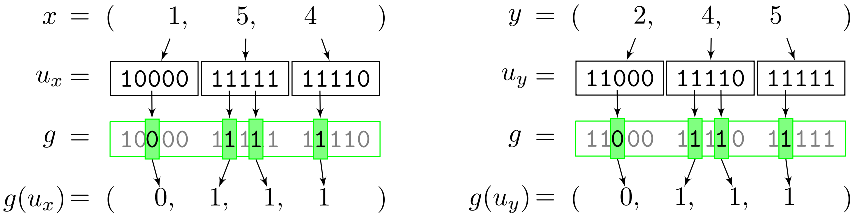

FIGURE 2. Exemplary locality sensitive hashing of x and y

In a first step the vector x has to be transformed into an *unary representation.* For instance the vector $x = (1, 5, 4)$ would become $u_x = (u(x_1) \circ u(x_2) \circ u(x_3)) = (10000\ 11111\ 11110)$ in unary notation by concatenating x_i ones followed by $C = \max(x) - x_i$ zeros. This step is necessary to apply the hash functions g_i, which map unary vectors to binary vectors of fixed length k, denoted as $g_i : u_x \rightarrow \{0,1\}^k$. Here, for each g_i, k indices are chosen at random of the size-fixed unary representation of x and are concatenated as follows: $g(u) = (u_{P(1)} \circ u_{P(2)} \circ \cdots \circ u_{P(k)})$, with set P, containing a random permutation of indices $\{1, \ldots, k\}$.

Each vector is organised in sets called *buckets.* A bucket, for example B_h, is uniquely identified by a hash value $h \in \{0,1\}^k$ and contains all pairwise collisions (i.e. potential similarities) of vectors which share the same LSH-value h.

[1] We use the term of locality in the sense of semantic similarity, rather than geographic location.

Taken the prior chosen value of $x = (1, 5, 4)$ and additionally $y = (2, 4, 5)$ with a single hash function g and an exemplary index permutation of $P = (3, 7, 9, 12)$ (implying $k = 4$) the resulting hash values would be $g(u_x) = g(u_y) = (0, 1, 1, 1)$ (see Fig. 2). Due to this hash collision, x and y belong to the same bucket $B_{(0111)} = \{x, y\}$, identified by the hash value of their collision ($ID(B_{(0111)}) = (0, 1, 1, 1)$). This would indicate that x and y are similar to each other.

LSH-heuristics are primarily used for an approximate – but quite efficient – computation of k-nearest-neighbor graphs in linear time [**KIW04**, p. 116], where the time complexity amounts to $O(nl|B|) < O(n^2)$. Beyond that, this technique can be used to locate similar data points to a given *prototype*, which can be seen as a search query to look for. The goal is to find data points (i.e. *candidates*), **near** a given prototype p, of a finite data set $\mathbb{X}^{\text{cand}} \subseteq \mathbb{X}$. Candidates s are *similar* to a search request p iif

$$s \in \mathbb{X}^{\text{cand}} \Leftrightarrow \exists g, g' \in \mathcal{H} : g(s) = g'(p) \vee g'(s) = g(p).$$

This means, that all s – for which at least one hash function of $\mathcal{H}$ leads to a collision with p – are candidates and therefore similar to p. From the set $\mathbb{X}^{\text{cand}}$, which is significant smaller than $\mathbb{X}$, data points worth further consideration can be selected using exact measuring methods due to the quite efficient elimination of irrelevant data in the previous step.

4. Locality Sensitive Data Model (LSDM)

We propose a three layer architecture based on LSH to address the problems as previously explained in the introduction, i.e., how to efficiently locate similar resources in distributed multi-agent environments, where the identification commonly takes place via cryptographic hash functions. We propose an extension to existing MAS-based resource management systems (e.g. [**VMCF05**], [**KFA07**]) with the intention of providing a highly flexible method to add similarity-based localization to other platforms and to reduce unnecessary complexity.

Architectual Design The actual extension consists of agents, managing lists of mutually similar resources. These agents, which we further call *locality-sensitive agents (LS-agents)*, are not only responsible for sharing their own resources, but also for managing similarity lists and keeping these up-to-date. As shown in Fig. 3, the LSDM extends a classic two-layer architecture, consisting of a resource and agent layer, with a third layer containing the LS-agents. The technique, presented in the previous Section 3 (finding similar vectors and organizing them in buckets by using LSH functions) will be modified in such a way that it can be built upon MAS.

The goal is to link reference specifications (search queries) q to sets of similar resources. To minimize the workload and to increase the availability of these sets, it is not advisable to entrust them to one single agent. Rather we distribute the buckets among available agents. This can be achieved by viewing buckets as resources and using the same cryptographic hash function f_X which is used to identify agents responsible for certain resources and to identify the resources themselves. If this hash function f_X (e.g. SHA-1) is applied to bucket-*ID*s, they can be treated like ordinary resources and the responsible agent can easily be determined by using the underlying distributed resource management system. For this assignment in detail, see Fig. 4. Here the LS-agents hold – beside their own ID – sets of similar resources S_h. Taking into account that storing whole resources in S_h is not desirable for reasons of data protection and possibly large resources, we only store their cryptographic hash values (which sufficiently identify resources) in S_h.

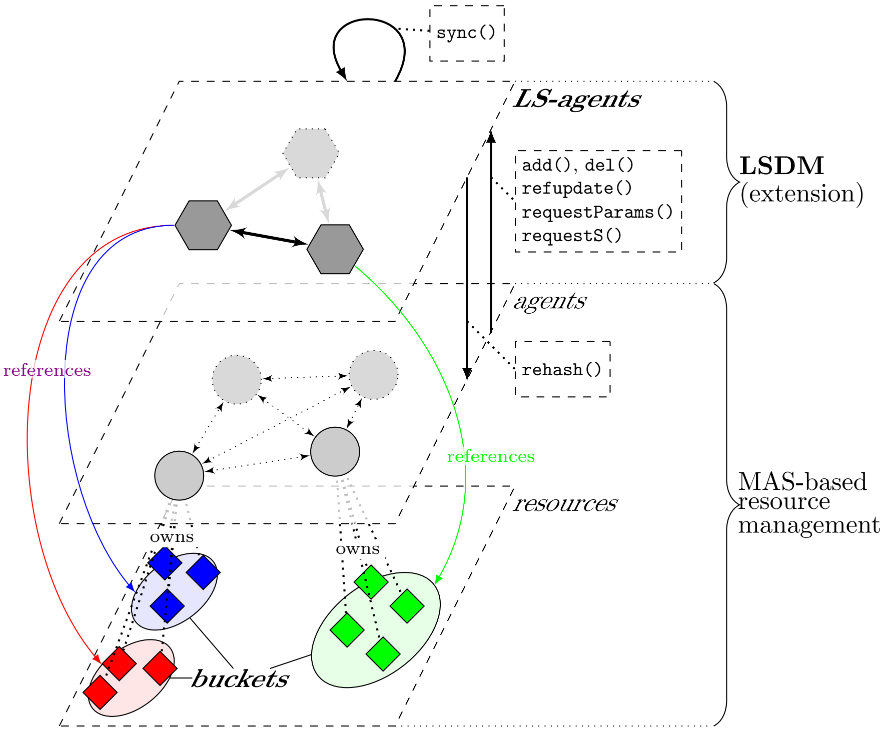

FIGURE 3. Three-layer architecture of LSDM

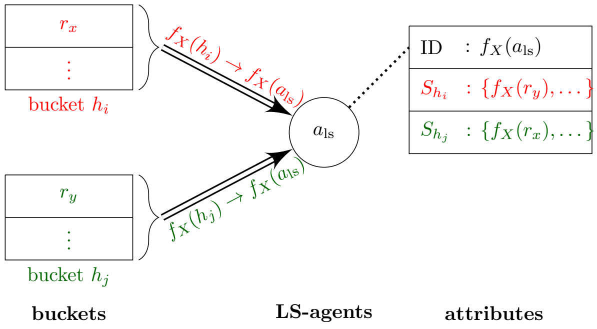
r_x
bucket h_i
r_y
bucket h_j
$f_X(h_i) \rightarrow f_X(a_{\text{ls}})$
$f_X(h_j) \rightarrow f_X(a_{\text{ls}})$
a_{ls}
ID : $f_X(a_{\text{ls}})$
S_{h_i} : $\{f_X(r_y), \dots\}$
S_{h_j} : $\{f_X(r_x), \dots\}$
buckets
LS-agents
attributes

FIGURE 4. Assignment of buckets to LS-agents by using a cryptographic hash function f_X

Identification of Similar Resources Assuming a reference specification $s \in \mathbb{R}^n$ exists, we can locate similar resources using the above described approach. At first we need to compute all LS-hash values h to determine bucket *ID*s, in which collisions (i.e. similar resources) can be found and put them into one set B_s: $B_s = \{h \mid h = g(s), \forall g \in \mathcal{H}\}$. Then we can identify the responsible agents by determining $ID_{\text{agent}}(h) = f_X(h)\ \forall h \in B_s$ and request their similarity sets S_{h_i} by using the underlying resource management system (e.g. issuing a `get`(ID$_{\text{agent}}(h)$) command). If we merge these sets S_{h_i} into one combined set $S = S_{h_1} \cup \dots \cup S_{h_n}$ we will get every *ID* of resources similar to the reference specification s.

Requirements to the LSDM Regarding the design of our data model, the following requirements are of great importance: 1. If information about similar objects is passed to the LSDM, it has to accept and process them in any case. 2. Similar resources (if they exist) to any given reference specification (i.e. prototype) have to be returned to the user. 3. Regularly modification of resources needs to be taken into account. 4. Resources, which are removed from the underlying data model, also needs to be removed from the LSDM. 5. In decentralized environments sign-outs or even unforeseen malfunctions of agents can not be ruled out. Therefore appropriate countermeasures need to be in place to prevent loss of information. These issues can be addressed as follows:

(1) ***Publication of Resources*** Initially every agent a, which publishes a resource r to the MAS, needs to have the necessary LSH parameters (i.e. $k, l, \mathcal{H}$), for which it issues the `requestParams()` command to one of the existing LS-agents. (Since the LSH parameters have to be the same throughout the LS-agents, it can ask any of them.) Then it can compute $h_i = g_i(r), \forall g_i \in \mathcal{H}$ $(i = \{1, \dots, |\mathcal{H}|\})$ to get a set of LSH-values h_i, which identify similarity lists, managed by LS-agents. These lists can now be requested from the MAS by viewing h_i as regular resources and thus get the responsible agents by $\mathrm{ID}_{\mathrm{LSagent}_i} = f_X(h_i)$. In the last step the publishing agent calls `add(`$\mathrm{ID}_{\mathrm{LSagent}_i}$`,`$h_i$`,`$f_X(r)$`)`, which tells $\mathrm{ID}_{\mathrm{LSagent}_i}$ to add the hash value of r to its similarity list S_{h_i}. Each LS-agent also stores, additionally to ID_r, the ID_a of its publisher, which is useful to push rehash requests to agents. (See Fig. 5 and *5. Exception Handling* below.)

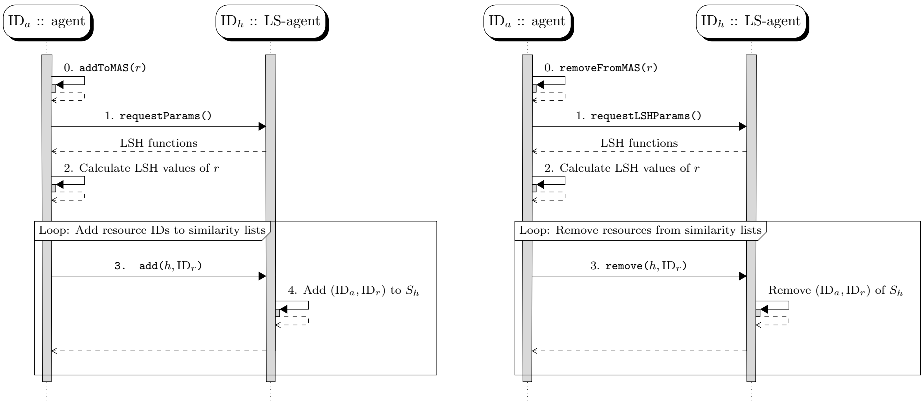
ID_a :: agent
ID_h :: LS-agent
0. addToMAS(r)
1. requestParams()
LSH functions
2. Calculate LSH values of r
Loop: Add resource IDs to similarity lists
3. add(h, ID_r)
4. Add (ID_a, ID_r) to S_h
ID_a :: agent
ID_h :: LS-agent
0. removeFromMAS(r)
1. requestLSHParams()
LSH functions
2. Calculate LSH values of r
Loop: Remove resources from similarity lists
3. remove(h, ID_r)
Remove (ID_a, ID_r) of S_h

FIGURE 5. The process of adding resources to LSDM (left) and removing them (right)

(2) ***Localisation of Similar Resources*** To find similar resources, according to a reference specification s, the agent requests the LSH parameters from any LS-agent via `requestParams()`. If it already received those, this step can be omitted. The agent needs to compute the set of LSH values B_s to get all LS-agents $\mathrm{ID}_{\mathrm{LSagent}_i} = f_X(h_i)$, with $h_i \in B_s$, responsible for bucket h_i. Thereupon the similarity lists S_{h_i} can easily be received by calling `requestS(`$\mathrm{ID}_{\mathrm{LSagent}_i}$`,`$h_i$`)` to the corresponding LS-agents. The actual resources, which are similar to s, are retrievable by their $ID \in S_{h_i}$ by using the methods of the underlying MAS.

(3) ***Modification of Resources*** If resources are modified ($r_{\mathrm{old}} \to r_{\mathrm{new}}$), it will have an impact on their similarities to other existing resources. Therefore the assignments of outdated ($f_X(r_{\mathrm{old}})$) and new ($f_X(r_{\mathrm{new}})$) resource *ID*s to similarity lists S need to be reviewed and – if necessary – updated. To accomplish this, the modifying agent calculates $B_{r_{\mathrm{old}}}$ and $B_{r_{\mathrm{new}}}$ (analogue to above) and tells every LS-agent derived from $B_{r_{\mathrm{old}}}$ to remove the similarity references (`del()`) and every LS-agent derived from $B_{r_{\mathrm{new}}}$ to add those (`add()`).

(4) ***Removal of Resources*** Resources r, which are removed from the MAS, need to be removed from the LSDM as well. Again, an agent needs to have the LSH parameters, which it can retrieve via `requestParams()`. It then computes all buckets which contain hash values $h_i = g_i(r), \forall g_i \in \mathcal{H}$, identifying similarity lists S_{h_i}. After that, the agent tells the responsible LS-agents to remove all references to ID_r from their similarity lists (`del()`). (See Fig. 5)

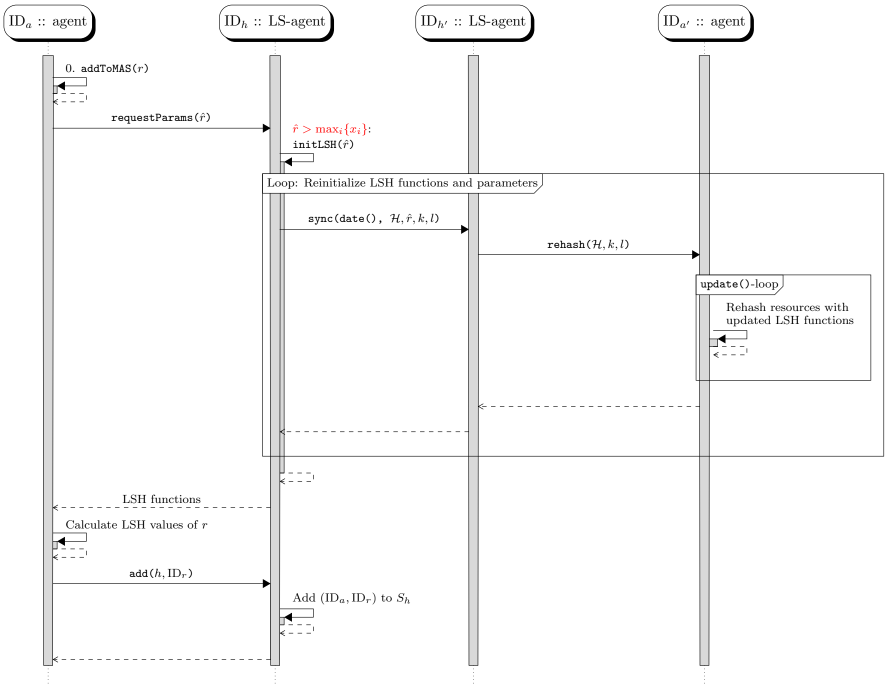
ID_a :: agent
ID_h :: LS-agent
$\mathrm{ID}_{h'}$ :: LS-agent
$\mathrm{ID}_{a'}$ :: agent
0. addToMAS(r)
requestParams($\hat{r}$)
$\hat{r} > \max_i\{x_i\}$:
initLSH($\hat{r}$)
Loop: Reinitialize LSH functions and parameters
sync(date(), $\mathcal{H}, \hat{r}, k, l$)
rehash($\mathcal{H}, k, l$)
update()-loop
Rehash resources with updated LSH functions
LSH functions
Calculate LSH values of r
add(h, ID_r)
Add $(\mathrm{ID}_a, \mathrm{ID}_r)$ to S_h

FIGURE 6. The process of adding resources if the maximal vector attribute is exceeded

(5) ***Exception Handling*** One aspect which arises from using LSH functions is, that once hash functions of $\mathcal{H}$ are defined, they constrain the size of possible vector attributes by $\max_{x \in \mathbb{X}}(x_i)$. If a new vector $y \in \mathbb{R}^n \setminus \mathbb{X}$ needs to be hashed, we have to *rehash* every existing resouce in $\mathbb{X}$ and rebuild and redistribute the buckets. In this case every LS-agent tells each agent a, which can be found in its similarity lists, to `rehash()` and republish their resources. (See Fig. 6)

Additional issues are malfunctioning or suddenly disconnecting LS-agents. Therefore, to prevent loss of similarity lists, we need to have backups among the agents of the MAS. We propose to have additional (backup) LS-agents to counteract these issues. Similar to [**SP03**] we can apply xor-metrics to LS-agent's *ID*s to determine *neighboring* agents and let them take over, if actual LS-agents are not available anymore.

5. Evaluation

The LSH technique underlying our data model is the central aspect of this paper, and its overall effectiveness depends on the accuracy of LSH's approximation. Therfore, we focus our evaluation on a qualitative analysis of our data model and on experimental comparisons between our approach and state-of-the-art clustering techniques on standard benchmark data. Here we want to show, that our approach can compete with well known clustering techniques and – moreover – prove its usefulness in decentralised environments, such as MAS, by providing a fair approximative classification of the available and shared resources in order to minimize the relevant similarity-search-space beforehand.

Qualitative Analysis For the distributed management of similarity information in multi-agent systems, our proposed data model LSDM – as part of a three layer architecture – can be an effective approach compared to centralized clustering of resources, by distributing the workload of clustering and communication over various agents. By introducing the concept of *locality sensitive hash functions*, we distributed the main workload of computing similarity information over the participating agents (i.e. collaborating partners, willing to share resources) and assigned the management of *similarity lists* to multiple LS-agents. Here the time complexity for each agent to LS-hash its resources amounts to $O(nl|B|)$ (see [**KIW04**, p. 116]), where n denotes the number of resources, l the number of LS-hash functions and $|B|$ the bucket size. Therefore this process is of linear computational complexity (see Section 3). As this step only needs to be done initially or if a rehashing is needed in exceptional cases (see Section 4), this approach has very little impact on the overall computation time of each agent. The management overhead of buckets, induced by the LS-agents is minimal, as these agents only act as *'yellow pages'* for search queries. This method also incorporates the idea of multi-agent systems, by providing a more fault tolerant and load balanced way of managing buckets compared to a centralized clustering instance and avoiding performance penalties induced by *bottlenecks* or *singe-point-of-failures.* Without the need for such an instance, our data model can be used in contexts with high volatility of available agents if precautions – in terms of backup LS-agents – are taken.

Experimental Setup and Results In our experimental scenario we measure the accuracy of finding similar elements from a database, containing disjunct classes of similar elements. As a benchmark data set for our evaluation, we chose the Wisconsin Diagnostic Breast Cancer database (WDBC), a well known benchmark set of clinical cases [**WSHM95**], which we transformed into integer-based vectors, in order to use it for our data model. The data set consists of 699 data points (but due to 16 incomplete results we could only use 683), each described as an 11-dimensional vector containing two identifying attributes (sample ID and classification). Therefore we used nine attributes as a data vector and the remaining two to assess the accuracy of our similarity approximation by using the Rand index.

In our experiment we only focused on the accuracy of LSH and measured the impact of different hash output lengths and number of used hash functions to the resulting accuracy of classification. We chose output lengths of 1 to 5, from 5 to 90 bits in steps of five and 1 to 20 hash functions. For each combination of number of hash functions and output length, we ran the LSH classification 100 times to obtain a good mean accuracy. In each iteration we randomised the hash functions to minimise errors due to bad functions. As shown we plotted the average accuracy against the number of used functions Fig. 7 and output length Fig. 8. (The omitted samples lied in between those shown and were removed for a better overview.)

What our results show is that the accuracy of LSH's resource classification depends on two important parameters: The number of used hash functions and the unary output length of them. Regarding the output length our experiment shows, that by increasing the output length towards the allowed maximum ($vmax_{wbc} = 90 = \text{len}(x) \max_{x \in \mathbb{X}}(x_i)$), the accuracy decreases continuously until it reaches a minimum of 0.46 (Fig. 8) for our chosen data set, which can also be seen in Fig. 7 for the output length of 90. By combining the results for both parameters, we measured an accuracy of 85.0% with a standard deviation of 0.7 for an – in our

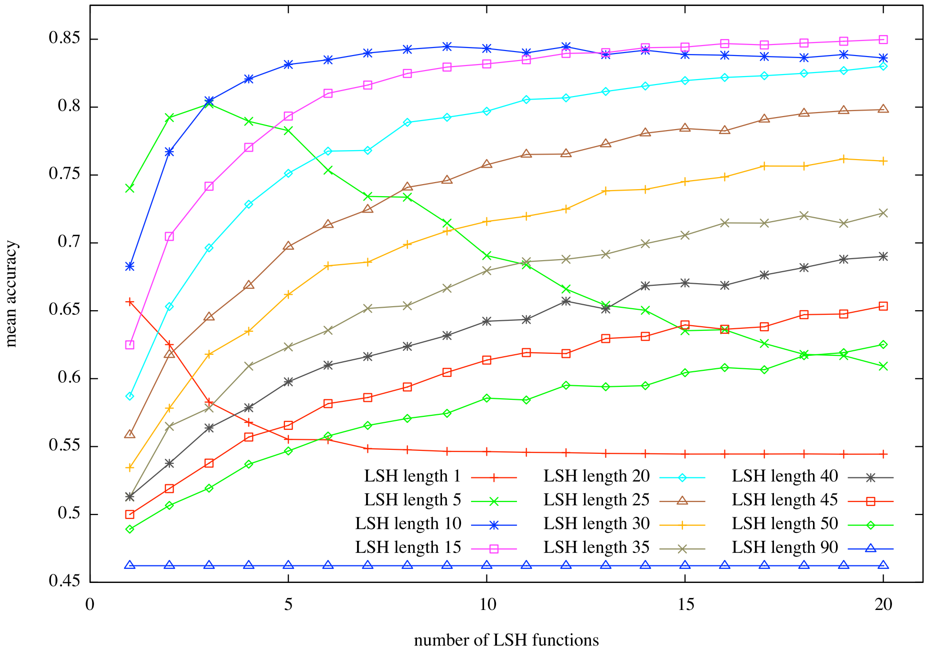

FIGURE 7. Accuracy of LSH against the number of used LS-functions

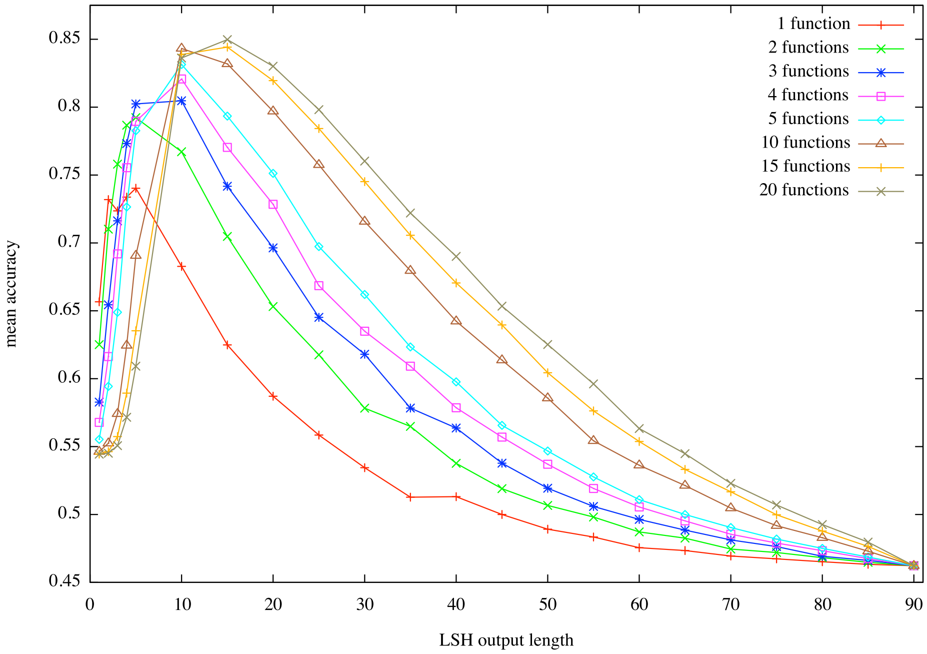

FIGURE 8. Accuracy of LSH against the output length of used LS-functions

case – optimal parameter choice of 20 hash functions with a length of 15. Regarding Fig. 7, one additional fact worth mentioning are the two curves for LSH lengths 1 and 5. Here we assume, that by increasing the number of hash functions, the amount of false-positives grows significantly faster compared to the other curves, due to the very short output lengths. Therefore the results for 1 and 5 can be treated as anomalies.

	LSH	k-Means [**HH07**]	Supervised Batch NG [**HH07**]	C4.5 decision tree [**HCSoRDoCS96**]
Accuracy in %	85.0	93.6	94.7	96.0
StdDev	0.7	0.8	0.8	–

TABLE 1. Accuracy of LSH compared to state-of-the-art techniques for WDBC classifications

In **??** we confronted our results for the WBCD with other, comparable, experiments (see [**HH07**], [**HCSoRDoCS96**]). As can be seen, heuristics like *k-Means*, *Neural Gas* and *C4.5* outperform our approach in terms of accuracy by roughly 10 – 20%. Here we would like to point out, that these heuristics operate centralized with complete knowledge of the available data set. Our data model in contrast, is able to compute bucket *ID*s without concrete knowledge of any existing resource, and purely relies on the approximation done by previously chosen LS-hash functions. Furthermore, every single agent only needs to know his own resources and does not need to exchange any information with other agents, except for buckets, containing cryptographic hashed values of resources. Due to the fact that the accuracy also considers elements that are similar but not located, it is possible to obtain them by increasing the number of hash functions and thus making it more likely for them to be found. Although this approach increases the set of false-positives, it can, by gradually extending $|\mathcal{H}|$, help to get more similar resources. Therefore, given that our main goal was to minimize the search-space for similar resources, an seemingly lower accuracy can significantly reduce the costs of comparing possible candidates, if the overhead, induced by non-similar resources, would still be orders of magnitude smaller than the whole search-space.

6. Conclusion and Outlook

The central aspect of the locality sensitive data model (LSDM) is the extension of an existing multiagent system (MAS), used in decentralized collaborated product development, to efficiently locate similar resources. The main contributions of this paper are twofold. First, we proposed and evaluated the incorporation of collision maximizing hash functions into existing agent-based resource management systems. Here the goal was to avoid a centralized approach, by using one single agent to manage sets of similar resources, but instead distribute these sets over multiple agents to minimize the management overhead, bottlenecks and single-point-of-failures. By using locality sensitive hash functions it is possible to compute the buckets, containing *ID*s of similar resources, in linear time and distribute them over LS-agents which are responsible for them.

Second, we showed in our experiments that that choose suitable LSH parameters (number of hash functions and output length) is important for achieving good results. Although we discovered that our approach is unable to achieve the same accuracy results as centralized state-of-the-art clustering and classification heuristics in terms of overall accuracy, we managed to reduce the search-space for similar resources significantly. Here further research is needed to conclude if the LSH parameter can be determined beforehand, as prior benchmarking and computation of optimal parameters is often not practical.

Our current approach faces several limitations: First we pointed out, that only vectors of $\mathbb{N}^n$ are suitable for LS-hashing. Therefore, as mentioned in Section 3, resources have to be transformed into integer vectors, by using specific characteristics of data objects (e.g. measurements) as vector attributes. Real-valued vectors for example could be easily transformed by using a fixed length of decimal places and multiplying them with a constant scalar value, but this would drastically limit the overall precision of the data set. Therefore it is reasonable to conduct further research on how to express product specifications as vectors or use different approaches in this context. Second, we have chosen the WDBC dataset for experimental evaluation in the absence of publicized benchmark sets of product models. While our method itself is general purpose, this may restrict generalizability of the experimental results to similarity search of product models. Research is needed towards creating appropriate benchmark datasets. Finally, we need to explore ways of further improving the accuracy of LSH-based search e.g., by modifying the random generation of LS-hash functions to weight more relevant vector attributes.

References

[DEG$^+$12] Peter Druschel, Eric Engineer, Romer Gil, Andreas Haeberlen, Jeff Hoye, Y. Charlie Hu, Sitaram Iyer, Andrew Ladd, Alan Mislove, Animesh Nandi, Ansley Post, Charlie Reis, Dan Sandler, Jim Stewart, Atul Singh, and Rong Mei Zhang, *Freepastry*, 7 2012.

[HCSoRDoCS96] H.J. Hamilton, N. Cercone, N. Shan, and University of Regina. Dept. of Computer Science, *RIAC: a rule induction algorithm based on approximate classification*, Tech. report, 1996.

[HH07] Barbara Hammer and Alexander Hasenfuss, *Relational neural gas*, Proceedings of the 30th annual German conference on Advances in Artificial Intelligence, KI '07, Springer Berlin / Heidelberg, 2007, pp. 190–204.

[KBC$^+$00] John Kubiatowicz, David Binde, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, , and Ben Zhao, *Oceanstore: An architecture for global-scale persistent storage*, Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)) (2000).

[KFA07] H. M. Kelash, H. M. Faheem, and M. Amoon, *A multiagent system for distributed systems management*, World Academy of Science, Engineering and Technology **11** (2007), pp. 91–96.

[KIW04] Hisashi Koga, Tetsuo Ishibashi, and Toshinori Watanabe, *Fast hierarchical clustering algorithm using locality-sensitive hashing*, Discovery Science, Lecture Notes in Computer Science, vol. 3245, Springer Berlin / Heidelberg, 2004, pp. 155–182.

[MM02] Petar Maymounkov and David Mazières, *Kademlia: A peer-to-peer information system based on the xor metric*, Revised Papers from the First International Workshop on Peer-to-Peer Systems (London, UK, UK), IPTPS '01, Springer-Verlag, 2002, pp. 53–65.

[SHM12] P. D. Stiefel, C. Hausknecht, and J. P. Müller, *Using ontologies to support decentral product development processes*, Agent-Based Technologies and Applications for Enterprise Interoperability (K. Fischer, J. P. Müller, and R. Levy, eds.), Lecture Notes in Business Information Processing (LNBIP), vol. 98, Springer-Verlag, 2012, pp. 114–129.

[Sie12] Siemens, *PLM, JT2Go, 2 Cylinder Engine (typical or "shattered" JT file)*, 7 2012.

[SOlTZ05] Yanfeng Shu, Beng Chin Ooi, Kian lee Tan, and Aoying Zhou, *Supporting multi-dimensional range queries in peer-to-peer systems*, FIFTH IEEE INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING, IEEE, 2005, pp. 173–180.

[SP03] Cristina Schmidt and Manish Parashar, *Flexible information discovery in decentralized distributed systems*, Proceedings of the 12th High Performance Distributed Computing (HPDC), IEEE Computer Society, 2003, pp. 226–235.

[VMCF05] P. Vilà, J. L. Marzo, E. Calle, and L. Fàbrega, *Multi-agent system co-ordination in a distributed network resource management scenario*, IEEE (2005).

[WSHM95] W. H. Wolberg, W. N. Street, D. M. Heisey, and O. L. Mangasarian, *Computer-derived nuclear features distinguish malignant from benign breast cytology*, Human Pathology 26 (1995), 792–796.

Clausthal University of Technology, Institute of Informatics, Julius-Albert-Str. 4, D-38678, Clausthal-Zellerfeld, Germany,, {malte.aschermann, joerg.mueller}@tu-clausthal.de