A Replication Overlay Assisted Resource Discovery Service for Federated Systems

Hao Yang, Fan Ye, Zhen Liu
IBM T. J. Watson Research Center
Email: {haoyang,fanye,zhenl}@us.ibm.com

***Abstract*—Federated systems have recently attracted much attention because they allow loosely coupled organizations to share resources for common benefits. However, discovering resources across administrative boundaries is challenging. Despite their willingness to share resources, many organizations prefer not to export their internal resource description to unfamiliar parties. While it is highly desirable to facilitate such *voluntary sharing*, the system also needs to resolve resource queries in an efficient manner. Unfortunately, none of the existing resource discovery designs, either hierarchical or DHT-based, can address these two challenges in the same time.**

In this paper, we present the design and evaluation of ROADS, a Replication Overlay Assisted resource Discovery Service for federated systems. In ROADS, the resource owners only export *summaries*, which are condensed representations of their resource records. These summaries are aggregated along a hierarchy and used to direct queries to appropriate resource owners. To improve its efficiency and resiliency, ROADS replicates the summaries using server overlays that enable "shortcuts" in query forwarding. We have implemented ROADS and evaluated its performance through extensive analysis and experiments. The results show that ROADS outperforms a DHT-based design with 1-2 orders of magnitude less overhead in update messages and 50% less query forwarding time.

I. Introduction

Federated systems have become increasingly popular nowadays because they allow loosely coupled systems within different organizations to share previously isolated resources for mutual benefits. Distributed System S [1] is an example where multiple stream processing sites, each owned and managed by a different organization, collaborate in performing complex processing tasks that are beyond the capabilities of any single site. However, discovering resource across the administrative boundaries is challenging. In particular, a participant's willingness to share resources by no means implies surrendering the control over its resources. Many organizations prefer *voluntary sharing* that preserves their autonomous management. Due to security and privacy concerns, they may decide not to disclose the detailed description of their resources to the public. They want to retain the final control over which resource records are returned for a given query. For example, a company may provide more resources to a business partner than arbitrary third parties.

This voluntary sharing poses new research challenges for resource discovery. Existing resource discovery designs, both hierarchical [2] and peer-to-peer [3], [4], [5], [6] approaches, do not lend themselves well to voluntary sharing. On one hand, the hierarchical designs suffer from inefficiency in multi-dimensional queries because the hierarchy is typically organized around a single attribute (e.g., a global namespace). To resolve a query with multiple attributes specified, the system needs to flood the query in a portion of the hierarchy. On the other hand, the peer-to-peer designs require each organization to export its original resource records to other participants. For example, in DHT-based schemes, the hashed ID of a resource record decides where it should be stored, which is typically not controlled by the original owner.

In this paper, we address the above challenges with ROADS, a Replication Overlay Assisted resource Discovery Service for federated systems. ROADS allows each participant to retain final control over its resources yet supports efficient multi-dimensional queries over dynamic resources. In ROADS, the participants form a hierarchy based on voluntary association among themselves, and each resource owner decides the extent and form of sharing by exporting resource data in coarse *summaries*. These summaries are aggregated in a bottom-up manner in the hierarchy to enable query forwarding along the reverse direction. To improve the system efficiency and resiliency, ROADS further uses replication overlays to enhance the hierarchy, where each node replicates the summaries of a few remote nodes and uses such "shortcuts" to speed up the query forwarding.

We have implemented ROADS and evaluated its performance using both analysis and experiments. Our results show that ROADS can effectively support voluntary resource sharing from diverse ownerships, and outperforms a peer-to-peer based resource discovery design [3] by reducing message and storage overhead by 1-2 orders of magnitude and decreasing query latency by more than 50%. To our best knowledge, ROADS is the first resource discovery design that can facilitate voluntary sharing and support efficient multi-dimensional search among diverse resource owners in a federated system.

The rest of the paper is organized as follows. Section II describes our federated system models and design constraints. Section III presents the design of ROADS in detail. We evaluate the performance of ROADS using analysis and experiments in Sections IV and V respectively, and discuss the related work in Section VI. Finally, Section VII concludes the paper.

II. System Models and Requirements

We consider a federated system that consists of loosely-coupled autonomous systems (called resource owners), man-

0190-3918/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPP.2008.61

aged by various organizations and falling under different administrative authorities. Each owner is willing to contribute a variety of resources, such as data sources, computing, memory and storage resources. These resources can be described by high-dimensional attribute-value pairs, and users submit multi-dimensional range queries to precisely specify their interests.

While the resource owners are willing to share resources for mutual benefits, they also prefer to retain the autonomous management of their resources. For example, based on who is requesting resources, it may decide which types of resources will be provided, thus presenting different "views" to different parties. Moreover, due to security and privacy concerns, many organizations do not want to disclose their internal resource description to third parties.

We believe that *voluntary sharing* is critical to address these practical issues and stimulate collaboration in a federated system. The resource discovery system should provide sufficient flexibility to resource owners such that each resource owner can independently make the most appropriate decisions about how to answer queries.

Different resource owners may use different attributes to describe their resources. For example, one may use "memory" to denote the available memory size while the other may use "RAM". This problem, known as schema mapping or integration, has been well studied in both the database [7] and grid computing communities [8]. We do not consider such issues and assume that all participants use a common schema for representing their resource records.

There are certainly many security and privacy issues in enabling resource sharing. For example, a resource owner may need to know the identity of the requesting party to decide which records to return, and some owner may misbehave. However, in this work, we assume that appropriate authentication mechanisms exist and participants are not malicious.

III. ROADS Design

In this section, we present the design of ROADS, a Replication Overlay Assisted resource Discovery Service for federated systems. ROADS can not only faciliate voluntary sharing but also support efficient search over dynamic resources. This is achieved through a hybrid approach, with both federated hierarchy and replication overlay techniques, that balances between the system flexibility and efficiency. In what follows, we will describe these mechanisms in detail.

A. Federated Hierarchy

As shown in Figure 1, ROADS provides resource discovery through servers that form a federated hierarchy. The resource data are propagated along the hierarchy, while the user queries are also resolved through the hierarchy. A server is a machine that can be provided by resource owners or third parties who can benefit from the sharing. We call them *server providers*.

Forming the Hierarchy The ROADS hierarchy is formed incrementally. To join the system, a new server needs to know one existing server in the hierarchy, say the root. This can be done by manual configuration or contacting any existing server. For the purpose of efficient search, it is desirable to maintain a relatively balanced hierarchy. As such, each server keeps track of the depth and total number of descendants for each child branch. This information can be derived and maintained from periodic bottom-up aggregation messages (see Section III-B).

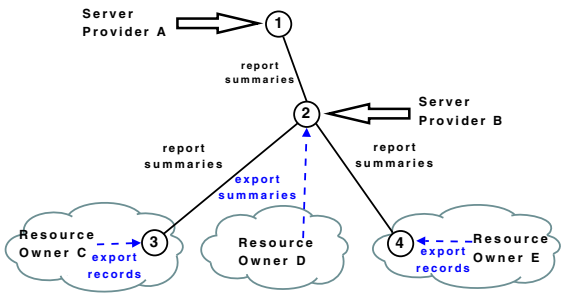

Fig. 1. An example of server hierarchy in ROADS. Server providers A and B provide servers 1 and 2 respectively; resource owners C and E host their own servers 3 and 4 respectively; resource owner D exports its resource summaries to server 2.

A new server queries the root server, and chooses the child whose branch has the least depth, or least number of descendants when depths are equal. It repeats the same process with the chosen child, until it finally reaches a server that is willing to accept it as a new child. It then attaches itself to this server. If it has reached a leaf server but still no one accepts it, it can backtrack to search other branches, until it finally finds a parent server. In the example (Figure 1), resource owner C hosts server 3; it chooses server 2 provided by another party B as server 3's parent. This way, server 3 becomes server 2's child in the hierarchy.

When deciding whether to accept a new child, a server may consider many factors, such as management and operational convenience, its current load, bandwidth utilization and network delay. For example, it may prefer servers in the same administrative domain, as it is easier to set up and maintain the association.

A resource owner can set up its own server or choose one server from the existing hierarchy. In either case, the resource owner exports its resource record data to the chosen server. We call this server the *attachment point* for this resource owner. The selection of attachment points follows a similar process as choosing parent server. Depending on whether a resource owner controls the attachment point or not, it can export resource records differently.

In the example shown in Figure 1, resource owner C hosts server 3 by itself. Since it has complete control over server 3, it can export detailed resource records on server 3. In contrast, resource owner D chooses server 2 provided by B, a different administrative ownership. Since resource owner D does not have control over B or sever 2, it exports summarized resource data. This way, it keeps the detailed resource records only to itself, but still allows its resources to be discovered.

Propagating Resource Data After receiving the detailed, or summarized resource record data from possibly multiple attached resource owners, a server stores them for use in evaluating queries. It also *aggregates* them into summaries and sends the results to its parent. Each server, after receiving

the summaries from its children, will again aggregate them, produce a branch summary and send it to the parent. Eventually the root server receives resource data through multiple levels of aggregation, originated from all resource owners.

Such bottom-up aggregation provides each server a coarse "*view*" of all resources exported under its branch. The root server has the global view of all available resources. These views will eventually guide the search down the hierarchy to the resource owners who have the matching data; they can then decide which resource records to return and in what formats.

Searching for Resources After the aggregation, the root server can serve requests for resources. A client can send the request in the form of a multi-dimensional query. The root searches through the summaries received from its children and discovers which branches have the requested data. It directs the client to further query those children, which then search their own data and direct the client further down appropriate branches of the hierarchy. Eventually the client reaches the resource owners or servers that have the detailed resource records. Now these resource owners can return specific resource records based on their local policies and preferences.

Hierarchy Maintenance The servers in the hierarchy may leave or fail, in which case the hierarchy must be properly maintained. Hierarchy maintenance has been well studied in other contexts, such as End-host Multicast, where a plethora of literature exists. We adopt a solution similar to [9], which can maintain a relatively balanced tree upon server departure or failures. A brief description is as follows.

Each parent and its child can exchange periodic heartbeat messages to detect failures. When several heartbeat messages are lost, one can assume the other end has failed. Each node also maintains a root path, containing all servers from the root to itself. This can be piggybacked on the heartbeat messages from a parent to its children. When a node leaves the hierarchy, it informs its parent P and its children. A child will try to rejoin the hierarchy starting from its grandparent P, found from its root path. If it cannot successfully find a new parent, it can start from one level up, the parent of its grandparent. Eventually it can start from the root again if needed.

Upon the departure or failure of a child, a parent will remove the summary and other state information associated with that child. The root path is also used to avoid loops: when a server joins a parent, it needs to ensure that it is not itself on the root path of the potential parent. Otherwise it should choose another parent. A special case is the failure of the root. The children of the root can elect one of them as the new root, using some simple rules such as the one with the smallest IP address or the lightest load. These children have to know each other, which can be achieved by the root sending a children list to each of them.

B. Bottom-up Aggregation

Aggregation Methods Aggregated summaries serve multiple purposes in ROADS. They are used to direct the search to appropriate resource owners in the hierarchy. They also reduce the amount of messages propagated in the hierarchy to make ROADS scalable. Different aggregation methods can be used in ROADS and we describe below a few typical ones.

As mentioned before, each resource is represented by a record with multiple attribute-value pairs. For example, a camera data source might be described as

```
{type=camera, encoding=MPEG2,
rate=100Kbps, resolution=640x480}
```

The attributes may be of different types such as integer, string, or categorical. Given a set of resource records, an aggregation method generates a summary, which is a condensed (and usually lossy) representation of the original records yet still supports query evaluation.

There are different ways to aggregate the records into summaries. A numeric attribute can be aggregated using a histogram consisting of multiple buckets of value ranges. Each bucket has a counter for how many values in this range are present. For categorical attributes, a set can be used to summarize all values in the given resource records. The set can directly enumerate all such values, which is acceptable if the number of distinct values is limited. More efficient data structures such as Bloom filters [10] or multi-resolution summarization [11] can also be used, as long as they compress data and support query evaluation.

Given a set of resource records, the values of each searchable attribute are aggregated, and the collection of such aggregated values becomes the summary of resource records. Such summaries can be further aggregated when they are propagated bottom-up in the hierarchy. For example, two histograms can be combined by adding their respective counters in each bucket. In this way, the amount of data is kept manageable for each server. We will analyze this in Section IV-B.

Evaluating Queries against Summaries After receiving a query, the server examines whether each child's summary matches the query. If so, it directs the client to further query that child. For example, upon receiving a query for

```
type=camera AND rate>150Kbps
AND encoding=MPEG2,
```

the server evaluates these three expressions against the summaries of the corresponding attributes. For example, the expression *rate>150Kbps* will be "true" when any of the buckets beyond 150 is non-empty in the histogram for data rates. An expression on categorical values (e.g. *encoding=MPEG2*) is "true" when the set contains the value (e.g., *"MPEG2"* is found in the set of encoding schemes). Finally the server obtains "true" or "false" results on each child's summary, and directs the client to query those children with results of "true".

Such coarse-grained summaries can utilize all queried dimensions to confine the search scope to only those servers with matching summaries. This avoids searching a wide range of servers, as in many other hierarchical based systems where the hierarchy is formed based on one special attribute (e.g., a unique name space).

Note that the process of exporting and aggregating resource data is repeated at each resource owner and server. Data and summaries are soft-state and have TTLs associated with them. This is because many resources are dynamic, thus we need to continuously update the corresponding resource records and summaries.

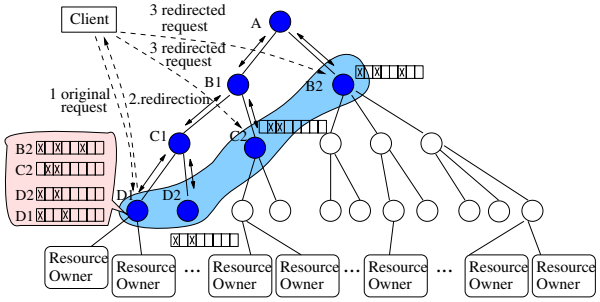

Fig. 2. Server D_1 has the summaries replicated from its sibling (D_2), its ancestors (C_1, B_1, A) and their siblings (C_2, B_2). The exchange of summaries between each parent and its children keeps the replicated data fresh. A client's search starts at D_1 and is redirected to C_2 and B_2, which then search their own descendants.

C. Replication Overlay

In the basic hierarchy, all queries must start from the root before they traverse the hierarchy. This increases the query resolving delay perceived by the clients, and the root becomes the bottleneck as well as a single point of failure. To address such issues, we use replication overlays on top of the hierarchy to improve the system availability and performance.

With replication overlays, each server replicates the branch summaries of *its siblings, its ancestors, and its ancestors' siblings* (in addition to storing the summaries from its children and directly attached owners). We choose such nodes such that each server stores summaries which combined together cover the whole hierarchy. As illustrated in Figure 2, server D_1 replicates the summary of its sibling D_2, its ancestor C_1's sibling C_2, B_1's sibling B_2, and its ancestors C_1, B_1, A. Thus each server has summaries for all portions of the hierarchy and the search can start at any server.

The above overlays require a server's branch summary be replicated at its descendants, its siblings and their descendants. This can be done by leveraging the existing hierarchy as follows. To replicate at its own descendants, a server simply propagates its summary down its own branch. To replicate at its siblings and their descendants, its parent can send its summary to its siblings, who can propagate the summary further to their descendants.

Upon receiving a query, a server evaluates the query against all summaries, including the replicated ones. When a match is found in a summary, the server directs the client to further query the corresponding server. For example (in Figure 2), the client first sends the request to D_1. After searching all summaries, D_1 finds that the summaries of C_2, B_2 match the request. It directs the client to send further queries to C_2, B_2, which can then search down their branches individually.

Such simple replication overlays provide several benefits. First, the bottleneck at the root is eliminated. Now search can start from any server in the hierarchy, not necessarily the root. Second, it reduces the query response time. The search can start from an attachment point server, which is usually lower in the hierarchy and closer to detailed resource records. If the attachment point server has eligible resource data, the client can obtain results immediately. Finally, it gives the client flexible control of the search scope. Each ancestor (or their siblings) of the starting server is one level higher in the hierarchy, providing more resources but requiring a longer search path. Based on the needs of how wide a range should be searched, the client can choose one or several branches to start its queries.

IV. Analysis

In this section, we analyze the performance of ROADS and compare it with two alternative approaches: a centralized repository and SWORD [3], a representative DHT-based resource discovery design.

With a central repository, all resource owners export their resource records to the repository, which answers queries by searching these records locally. With SWORD, the servers are organized into multiple DHT rings[1], one for each searchable attribute. The hash function in use preserves data locality and maps a range of values to a continuous segment on the ring. A resource owner registers its resource records in *all* rings, i.e., each record is replicated multiple times (one copy for each searchable attribute). A multi-dimensional range query is resolved in *one* ring and forwarded to the segment that corresponds to the queried range for the attribute associated with this ring. Servers within this segment search their local records against all attributes in the query, and then return any matching records.

Note that neither SWORD nor the centralized repository supports voluntary sharing because they require the exporting of original resource records. However, we are still interested in how ROADS compares to them in terms of system performance. To briefly highlight, our analysis finds that ROADS has 1-2 orders of magnitude less resource update and storage overhead than these alternative designs.

A. Notations

We use the following notations in the analysis. There are N resource owners in the system, each holding K resource records. Each record has r numeric attributes taking values from unit range. The size of an attribute's value is 1, thus each record has a size of r. The summary uses histogram and has m buckets for each attribute. Regardless of the number of resource records N, it always has a constant size of mr. Moreover, the records are dynamic and each record changes every t_r seconds, while a summary changes every t_s seconds. When a record has a changed attribute, the summary may stay the same, e.g., when the changed value remains within the same bucket in the histogram. Thus, typically we have $t_s << t_r$. The user query has q attributes, each specifying a range with length α ($\alpha < 1$). There are n servers and they form a balanced hierarchy of $L+1$ levels in ROADS. Each parent has k children. The resources are exported to leaf servers.

B. Results

Resource Update Overhead We first analyze the total message overhead for resource updates in different approaches.

[1]All these rings are connected and can be viewed as multiple sub-rings in a single ring.

As dynamic resources are updated periodically, we use the message overhead per second as the metric.

In ROADS, the resource update overhead is incurred by exporting summaries to attachment point servers, bottom-up summary aggregation and top-down summary replication. As the N resource owners each exports one summary of size rm, the summary exporting overhead is $O(rmN)$. The bottom-up aggregation incurs $n-1$ messages, one over each link, and the top-down replication incurs $O(knlogn)$ messages. Each of these messages contains one summary and has a size of rm. Note that these updates are sent every t_s second, thus the per-second resource update overhead is

$$T_{ROADS} = O(\frac{rm(N+knlogn)}{t_s}) \quad (1)$$

In SWORD, however, the resource update overhead is incurred by exporting original resource records to the DHT rings. Note that each record is stored in r servers, one copy for each ring, and it takes $O(logn)$ steps to route the record in each ring. The overall message overhead to store KN records, each having a size of r, is $O(r^2KNlogn)$. This process repeats every t_r seconds. Therefore, the per-second resource update overhead in SWORD is

$$T_{SWORD} = O(\frac{r^2KNlogn}{t_r}) \quad (2)$$

With a central repository, each resource owner directly exports its resource records to the repository. As such, every t_r seconds, it takes $O(KN)$ messages to export the KN records. Thus the per-second resource update overhead is

$$T_{Central} = \frac{rKN}{t_r} \quad (3)$$

To get a sense out of the equations, we consider some specific parameter settings as follows. Each record has $r = 25$ attributes; a histogram contains $m = 100$ buckets; there are $k = 5$ children for each ROADS server in a $L = 4$ level hierarchy (156 servers in total); the relative update frequency $t_r/t_s = 0.1$ (i.e., summary changes one order of magnitude slower than records). We calculate their overhead and find that 1) ROADS has about 1-2 orders of magnitudes less overhead than SWORD (confirmed by simulation results in Figure 4). This is because the summary size is orders of magnitude smaller than the original resource records, and it remains constant regardless of the number of records; 2) Both SWORD and the central repository have linearly increasing overhead w.r.t the number of records (KN), while SWORD has an overhead $r \log n$ times higher than the central repository. This is because in SWORD, each record is replicated r times, one copy for each attribute.

Summary Maintenance Overhead The message overhead for summary maintenance is unique to ROADS. For a level-i node, the overhead is about $O(k^2 i)$. Thus, the total summary maintenance overhead in the worst case is

$$S^{worst}_{ROADS} = O(k^2 logn)/t_s \quad (4)$$

For a $L = 7$ level hierarchy where each parent has $k = 5$ children, the largest per-node overhead is about 150 summary messages per t_s seconds (on the order of several minutes at least). In other words, each node only sends a few summaries per second, which is very small. This demonstrates the efficiency of the summary replication. Even with a large hierarchy, the summary maintenance does not incur much wide-area traffic.

	ROADS	SWORD	Central
Storage Overhead	$rmk(i+1)$	r^2KN/n	rKN
Exemplary Value	2×10^5	6.4×10^8	10^9

TABLE I
STORAGE OVERHEAD COMPARISON AMONG ROADS, SWORD AND THE CENTRAL REPOSITORY

Storage Overhead The data stored by ROADS servers are the summaries, while in SWORD and the central repository they are the original resource records. For ROADS, a level-i node maintains k summaries from its children, ki summaries from its ancestors and ancestors' siblings. For SWORD, all the KN records are stored in each ring of n/r servers. Assume each server receives the same amount, it is rKN/n records per server. For the central repository, it is KN records. Table I summarizes their overheads and give exemplary values[2], assuming $N = 10^3$ resource owners, $K = 10^4$ records, and other numbers are the same as used before. We can see that ROADS has orders of magnitude lower storage overhead than the other two. Again, this shows the benefit of using summaries whose size is much smaller and remains constant regardless of the number of records.

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of ROADS and compare to SWORD using simulations. The metrics of our interest include: 1) *query latency*, defined as the time from the client initiating a query to the query reaching the last server it needs to contact; 2) *resource update overhead*, defined as the total number of bytes sent for updating the resource records or summaries; and 3) *query message overhead*, defined as the total number of bytes sent for forwarding the queries. We also evaluate how data distribution and node degree, two factors that apply only to ROADS, affect its performance. Finally we report some benchmarking results obtained from a prototype ROADS system.

By default, our simulations use 320 nodes, each having 500 records. Each record has 16 attributes, with 4 different types of distribution: uniform (uniformly distributed in [0,1]), range (uniformly distributed in ranges of length 0.5), Gaussian and Pareto (scaled and truncated into [0,1]). There are 500 queries, each having 6 dimensions: two on uniform attributes, two on range attributes, one each on Gaussian and Pareto attributes. Each query dimension specifies a range of length 0.25, and each query is initiated from a randomly chosen node. We use the 5-dimensional synthesized coordinate system in [12] to simulate the network latency between any given pair of nodes over the Internet. In the ROADS hierarchy, a node has a maximum of 8 children. The histogram has 1000 buckets for each attribute. The results are averaged over 10 runs.

[2]It is the worst case for ROADS, which happens at the leaf nodes.

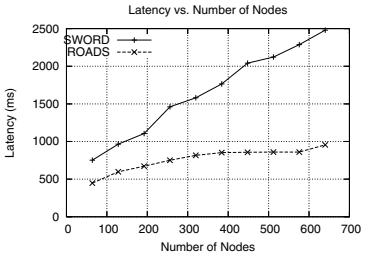

Fig. 3. Query resolving latency as a function of the number of nodes. The latency increases logarithmically in ROADS but linearly in SWORD; ROADS has about 50%∼60% less query latency than SWORD.

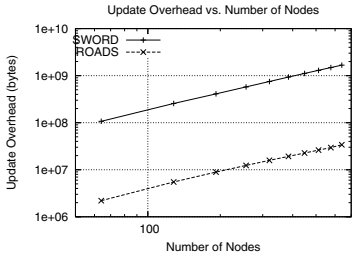

Fig. 4. Update message overhead as a function of the number of nodes. ROADS has two orders of magnitude less update overhead than SWORDS due to the use of condensed summary.

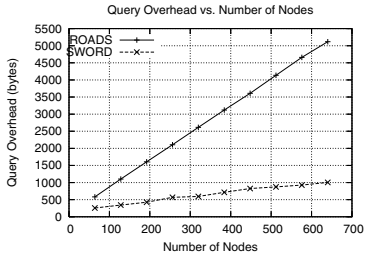

Fig. 5. Query message overhead as a function of the number of nodes. ROADS has 2∼5 times higher query overhead than SWORD, because ROADS has to visit more servers due to voluntary sharing.

A. Comparison with Sword

Impact of System Size We first vary the number of nodes from 64 to 640, in increments of 64, and measure its impact on query latency and message overhead. The query latency results are shown in Figure 3, from which we can make two observations. First, ROADS has much less (40-50%) query latency than SWORD. This is because ROADS can search multiple branches in parallel and the latency is determined by the number of levels in the hierarchy, which is 3∼5 in the simulations. Second, the query latency increases logarithmically for ROADS, while linearly for SWORD. This is because the query in SWORD sequentially traverses nodes in the matching segment, the size of which is proportion to the total number of nodes for a fixed query selectivity. The small jump for ROADS at 640 nodes is due to the increase of hierarchy level from 4 to 5.

The results of resource update overhead and query overhead are shown in Figure 4 (in log scale) and Figure 5 (in linear scale) respectively. We can see that ROADS and SWORD make different tradeoffs between these two overheads. On the one hand, ROADS reduces the update overhead by nearly two orders of magnitude (shown in Figure 4) due to the use of highly condensed summaries. On the other hand, ROADS incurs 2∼5 times higher query message overhead (shown in Figure 5). This is because for voluntary sharing purpose, each ROADS server retains the original resource records and thus the query has to be forwarded to all nodes that have matching records. In contrast, SWORD can hash the matching records to a relatively smaller number of servers. Nevertheless, the orders of magnitude reduction in update overhead dominates the query overhead. Thus ROADS still incurs much less total message overhead.

Impact of Query Dimensionality Next we vary the number of dimensions in the query from 2 to 8 and evaluate how ROADS and SWORD are affected. Since the update overhead does not depend on external queries, ROADS still has about two orders of magnitude less update overhead. Figures 6 and 7 show the query latency and query overhead results respectively, as functions of query dimensionality. We can see that the latency in ROADS decreases by roughly 40% as the number of query dimensions increases from 2 to 8. This is because ROADS utilizes all dimensions in the query to confine the search scope. Only branches with summaries matching all queried dimensions will be further searched, and this number decreases when the query has more dimensions. In contrast, SWORD only uses one dimension in the search. Thus its query latency remains largely the same (shown in Figure 6), even though more dimensions are available to confine search scope.

As shown in Figure 7, SWORD has linearly increasing query overhead as the query dimensionality grows. This is simply because the size of query messages becomes larger. ROADS shows an initial decrease in query overhead, because less query messages are sent as the search scope is confined by the increased query dimensionality. However, as the query message size further increases, the query overhead increases again because the reduction of search scope flattens out. In summary, ROADS can take advantage of the high dimensionality in queries, but SWORD cannot.

Impact of the Number of Records We also increase the number of records per node from 50 to 500 and measure how it affects ROADS and SWORD. When other parameters (number of nodes, query dimensionality, etc.) remain the same and only per-node record number increases, a query is forwarded along the same path in SWORD, and almost the same servers in ROADS. Thus query latency and overhead remain unchanged in both cases. We depict the update overheads in Figure 8, which again shows ROADS has orders of magnitude less overhead (similar to Figure 4). Due to the use of constant-size summaries, the update overhead in ROADS remains constant when each node stores more records. In contrast, Sword exports original records and thus its update overhead grows linearly with more per-node records. As such, the benefits of ROADS become more apparent when servers have larger number of records.

B. ROADS-Specific Settings

In this subsection, we further evaluate the impact of several system issues that are unique to the ROADS design.

Data Distribution How data is distributed among different servers in ROADS affects which servers have matching resource records, thus which servers will be contacted eventually. In general, the more heterogeneous the servers' local resource records are, the less query overhead ROADS has.

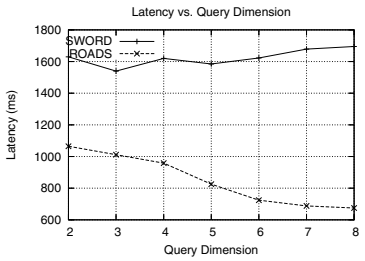

Fig. 6. Latency as a function of query dimensionality. ROADS utilizes all dimensions in a query to confine its search scope, thus reducing latency for high-dimensional queries.

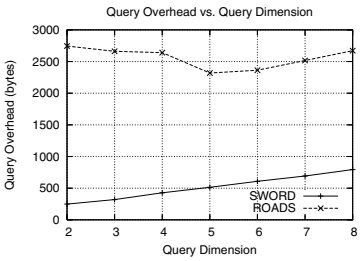

Fig. 7. Query overhead as a function of query dimensionality. Due to smaller search scope, ROADS initially has less overhead when query dimensionality increases.

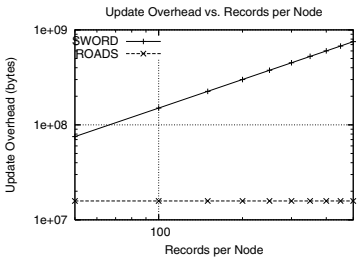

Fig. 8. Update overhead as a function of per-node record number. ROADS has constant overhead due to the use of fixed-size summary; Sword has linearly increasing overhead.

We measure the impact of data distinction by using an *overlap factor*, O_f, to control the extent of overlapping between the resource data at different servers. Specifically, for each of the first 8 attributes, we let the resource data of each server distribute within a range of length $O_f/320$, randomly located within [0,1]. Note that there are 320 nodes in total, and we vary O_f from 1 to 12. The query latency results are shown in Figure 9. We can see that as O_f increases from 1 to 12, the latency increases slightly from 810 to 860 ms (about 8%). This is because more servers have matching records when their data exhibit larger overlaps. We found a similar increase (about 10%) in the query overhead, but the update overhead is not affected (Figures not shown due to space limit).

Node Degree The node degree affects the depth of the hierarchy, thus how many hops a query traverses before reaching the leaf nodes. We vary node degree from 4 to 12 and, as Figure 10 shows, the query latency decreases from 1000 ms to 650 ms. Such latency reduction is mainly because the hierarchy becomes "flatter", thus a query is forwarded to leaf nodes in fewer hops. For the same reason, we also observed that the query overhead dropped from 3500 to 2000 bytes (Figure not shown due to space limit).

Prototype Benchmarking We have implemented a ROADS prototype using Java and benchmarked its performance on a testbed. Since metrics such as message or state overhead can be analyzed or simulated relatively accurately, we focus on the *total response time*, which is the time for a client to receive all matching records after it sends out a query. This response time is incurred not only by the network latency but also by the time the servers take to search through their local records and return all matching results. Note that the latter part is difficult to simulate or analyze because it may involve a backend database storing many records. In our previous simulations, the query latency includes only the time for query forwarding, but not the server processing or result returning time.

Our testbed has a cluster of Linux machines with Xeon 3.4GHz CPU and 6 GB memory. Each server maintains a DB2 database to emulate the attached resource stores, and uses JDBC interface to query this database for specific resource records or to generate summaries. We store 200K resource records at each server, and each record has 120 attributes, including integer, double, timestamp, string, categorical types. We populate these databases using both synthesized and real data collected from the Distributed System S platform [1].

During the experiments, we feed the system with a large set of multi-dimensional search requests. These queries are grouped based on their selectivity, defined as the percentage of resource records that match the query. There are 6 groups in total, with a selectivity of 0.01%, 0.03%, 0.1%, 0.3%, 1%, and 3%, respectively. Each group has 200 queries.

Figure 11 shows the average and 90 percentile total response time as a function of query selectivity. The centralized repository is faster when the selectivity is low. This is because only one round of query and reply exchange is needed with a central repository. As selectivity increases, however, the response time of ROADS becomes comparable to (with 1% selectivity), or even better than (with 3% selectivity), that of a central repository. This is because the response time is dominated by the time to retrieve and return resource records. Multiple ROADS servers can do this in parallel, thus outperform the central repository using a single server. We also see that ROADS has response time of about 1000 ms when the selectivity is below 0.3%. This is consistent with the previous simulation results of roughly 800 ms query latency time.

VI. Related Work

The existing resource discovery systems can be roughly categorized into either hierarchical or flat, DHT-based designs [13]. A hierarchical system, such as LDAP [2], is typically organized based on a globally agreed dimension (e.g., a namespace), along which the resource records are divided top-down and delegated to participating parties. Queries are resolved based on this special dimension, and the system cannot leverage other attributes in a query to confine the search scope. In contrast, the federated hierarchy in ROADS is formed incrementally through voluntary association among servers, which is critical to facilitate voluntary sharing in federated systems. The bottom-up aggregation of all searchable attributes into summaries also allows ROADS to better confine the search, because all attributes in a query are examined when it is forwarded within the hierarchy.

The DHT-based resource discovery systems, developed for Grid computing [4], [6] or other contexts [3], [14], organize the servers into P2P networks and resolve the queries using various P2P routing protocols. Such systems have both incentive and performance drawbacks in the voluntary sharing

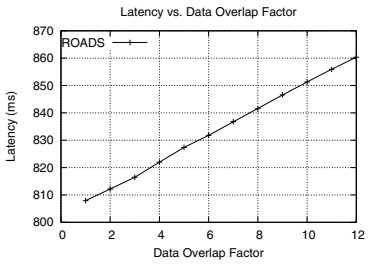

Fig. 9. Latency as a function of data overlap factor. As the degree of data overlap among servers increases, more servers have matching records and the resulting query latency increases slightly (by up to 8%).

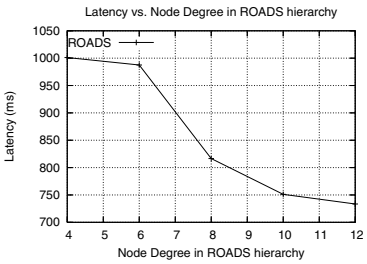

Fig. 10. Latency as a function of node degree. As node degree increases, the hierarchy depth decreases. As such, the query latency drops because less servers are contacted before the query reaches the leaf servers.

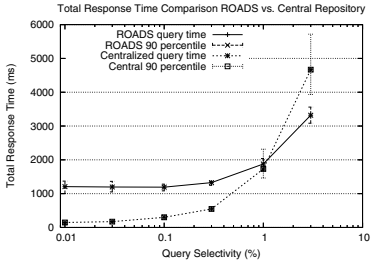

Fig. 11. Total response time as a function of query selectivity. Centralized repository outperforms ROADS when queries have few matching data. As query selectivity increases, ROADS has comparable response time.

context. First, the owner has no control over where his resource records are stored, because it is decided by a hash function. It is possible that an owner's records are hashed to undesirable parties, such as arbitrary strangers. Also, the only form of sharing is to export the detailed resource records. These conflict with the owners' desire to control how queries are answered. Secondly, to handle multi-dimensional queries, the DHT-based system typically builds multiple rings, one for each dimension [15], [16]. For one particular query, however, the search is performed only in one ring. As shown in our evaluation, this incurs significant data replication overhead with a high volume of dynamic resources. In contrast, ROADS replicates only resource summaries, which are much smaller and much less dynamic than the original records. It also utilizes all dimensions in query forwarding: only those nodes whose summaries satisfy all dimensions are further queried.

Several content-based publish/subscribe systems [17], [18] also use summaries, such as set union and Bloom Filters, in routing events to relevant brokers. However, in these systems, the summaries are the aggregation of subscriptions (i.e., queries), and every broker knows the summaries of all other brokers. In contrast, ROADS uses data summaries to facilitate voluntary sharing, and each server only knows the summaries of $O(\log N)$ other servers (i.e., its children, its ancestors and its ancestors' siblings) in the hierarchy.

VII. Conclusions

Discovering resources across loosely coupled yet autonomously managed organizations presents unique challenges in federated systems. To this end, we have presented the design and implementation of ROADS, a resource discovery service that can facilicate voluntary sharing. ROADS preserves the autonomous control of resource owners through a flexible federated hierarchy and the use of coarse resource summaries. It also provides efficient support for high-dimensional range queries using summary replication overlays. Our extensive analysis and experiments have demonstrated the scalability and efficiency of ROADS in terms of both message overhead and query latency.

In a broader context, there are many other issues, such as security, load balancing and churns, that a resource discovery system must address. We plan to explore various techniques to enhance the ROADS design along these dimensions in our future research.

References

[1] M. Branson, F. Douglis, B. Fawcett, Z. Liu, A. Riabov, and F. Ye, "CLASP: Collaborating, autonomous stream processing systems," in *Proc. ACM Middleware*, 2007.

[2] W. Yeong, T. Howes, and S. Kille, "Lightweight directory access protocol," *RFC1777*, 1995.

[3] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and implementation tradeoffs for wide-area resource discovery," in *Proc. IEEE HPDC*, 2005.

[4] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Proc. IEEE HPDC*, 2001.

[5] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery," in *Proc. International Conference on Pervasive Computing*, 2002.

[6] D. Spence and T. Harris, "XenoSearch: Distributed resource discovery in the XenoServer open platform," in *Proc. IEEE HPDC*, 2003.

[7] D. Embley, L. Xu, and Y. Ding, "Automatic direct and indirect schema mapping: Experiences and lessons learned," *SIGMOD Record*, vol. 33, no. 4, Dec 2004.

[8] M. Aktas, "Information Federation in Grid Information Services," Computer Science Department, Indiana University," Ph.D. Dissertation, 2007.

[9] B. Zhang, W. Wang, S. Jamin, D. Massey, and L. Zhang, "Universal IP multicast delivery," *Computer Networks*, July 2005.

[10] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, 1970.

[11] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann, "Multi-resolution storage and search in sensor networks," *ACM Transactions on Storage*, August 2005.

[12] B. Zhang, T. E. Ng, A. Nandi, R. Reidi, P. Druschel, and G. Wang, "Measurement-based analysis, modeling, and synthesis of the Internet delay space," in *ACM/USENIX Internet Measurement Conference*, 2006.

[13] R. Ranjan, A. Harwood, and R. Buyya, "A study on peer-to-peer based discovery of grid resource information," *IEEE Communication Surveys and Tutorials*, 2007.

[14] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next neneration name service for the Internet," in *Proc. ACM SIGCOMM*, 2004.

[15] A. Bharambe, M. Agrawal, , and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," in *Proc. ACM SIGCOMM*, 2004.

[16] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-peer information retrieval using self-organizing semantic overlay networks," in *Proc. ACM SIGCOMM*, 2003.

[17] Y.-M. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. J. Wang, "Subscription partitioning and routing in content-based publish/subscribe networks," in *Proc. DISC*, 2002.

[18] Y.-M. Wang, L. Qiu, C. Verbowski, D. Achlioptas, G. Das, and P. Larson, "Summary-based routing for content-based event distribution networks," *ACM Computer Communication Review (CCR)*, Oct 2004.