A Query Language for Network Search

Misbah Uddin, Rolf Stadler
ACCESS Linnaeus Center
KTH Royal Institute of Technology
Email: {ahmmud,stadler}@kth.se

Alexander Clemm
Cisco Systems
San Jose, California, USA
Email: alex@cisco.com

***Abstract*—Network search makes operational data available in real-time to management applications. In contrast to traditional monitoring, neither the data location nor the data format needs to be known to the invoking process, which simplifies application development, but requires an efficient search plane inside the managed system. This paper presents a query language for network search and discusses how search queries can be executed in a networked system. The search space consists of named objects that are modeled as sets of attribute-value pairs. The data model is more general than the relational model, and the query language is more expressive than relational calculus. The paper shows that distributed query processing can be performed using an echo algorithm and that name resolution can be embedded in query processing. Finally, two use cases for network search are presented, one in networking and one in cloud computing, the latter backed up by a prototype implementation.**

***Keywords*—Network search, management paradigms, distributed management, name resolution.**

1. Introduction

Network search—or search in networked systems—can be understood in three ways. First, as a generalization of monitoring whereby the monitoring data is retrieved by characterizing its content in simple terms, without giving location or detailed structure of the data. Second, it can be understood as "googling the network" for operational data, in analogy to "googling the web" for content. Third, network search can be seen as a capability that views the network as a giant database of operational and configuration data, which can be queried through a database-like interface.

In this paper, we follow the database interpretation and develop further the concept of network search, which we motivated and introduced in [42]. Specifically, we view network data as objects with a simple structure: an object has a (globally unique) name, a type and a variable number of additional attribute-value pairs. Objects are linked through joint attribute-value pairs through which associations can be expressed and discovered. We introduce a language to express search queries. It turns out that processing search queries can be performed through similar techniques as query processing in distributed (relational) database systems; we propose a tree protocol that dynamically creates spanning trees inside the networked system and incrementally aggregates the partial search results, which are sent from the leafs of the tree towards the root. Object name resolution is embedded in a "natural way" in query processing.

While some database concepts help in engineering a network search system, there are clear differences between querying a traditional distributed database and performing search in a networked system. First, a search result may not be an exact match, but only 'close enough' to be returned by a query. Second, similar to web search, search results are ranked, according to how closely a specific object in the result matches the query, the search history, etc.

We believe that an important application area for network search will be capturing and tracing dynamic service behavior across time and space. For instance, network search can be used to identify and trace media streams associated with a videoconference across the nodes of a network, or to a find the set of virtual machines associated with a particular application in a server cluster. The paper will provide more details about these use cases. Such functionality can, of course, be "hardcoded" beforehand in specialized protocols; network search, however, allows us to dynamically introduce such capabilities into a networked system.

The paper is organized as follows. Section 2 discusses related work. The overall framework for a network search system is shown in Section 3. Section 4 contains the proposed data model. Section 5 describes the query language, and Section 6 explains how search queries can be processed in a distributed fashion in a networked system. Section 7 discusses two use cases in some detail, and Section 8 reviews the paper's contribution and presents future work.

2. Related Work

There are two main research areas that relate to the topics adressed in this paper: web search and its evolution and query processing in large networked systems. For further research areas related to network search, see related works section in [42].

Web search, as exemplified by Google's search engine, is performed by matching keywords against the universe of web pages [31]. Search results are presented as ranked lists of links to web pages, which can be accessed through a web browser. Matching is performed on a distributed inverted index of the web pages, using a dedicated search infrastructure outside the web. The index database is populated through so-called crawlers, which continuously navigate the web following hyperlinks, i.e., the links between web pages [17]. The rank of a matched page is determined by several metrics, including keyword-relevance score [19], which measures the relevance of the keywords for the page content, the authority score, which measures the level of connectedness of the page in the hyperlink graph [22] [17], and page statistics [10].

As in web search, the search space in network search is very large. Objects in network search tend to be more dynamic and can have short life time than web pages, which requires a different architecture for network search, that allows for query processing close to the data (see Section 3). The concept of ranking is important in both areas. It has been very well developed in web search, while in our work on network search, it is part of our future plans.

Web search has evolved to include so-called live search on news, blogs, microblogs, etc. Live content expires at a faster rate than regular web content [12] [41]. To maintain the index structure for search, new techniques have been developed, including partial indexing [18], RSS feeds [7] [12], adaptive periodic crawling [19]. In addition to the metrics mentioned above, the freshness of the data is a determining factor in the ranking of search results [12] [21] [18]. Note though that the overall architecture of web search has not changed with the introduction of live search. In particular, matching is performed in a dedicated infrastructure outside the web.

A second innovation in web search has been annotating web pages with names of real-world or abstract entities, which has been initiated by Wikipedia [8]. The concept of annotated web has triggered research activities into searching for relationships between entities in web pages, for example, the fact that person x collaborates with person y. Two approaches are being pursued. First, an approach whereby such relationships are explicitly defined and then queried [25]; second, an approach where relationships are discovered through language analysis [27].

We believe that the concept of discovering relationships between objects is also important to network search. The language introduced in this paper allows to discover links between objects which have not been explicity declared.

A recent focus of research has been search in web-based social networks. Social networks have concepts like friends and groups and allow actions such as sharing, liking, recommending etc., also known as social tagging [45]. Search in this context are guided by two principles. It is user-centric [13] [26] and it uses social graphs as a means to guide the search process in a vast search space [43] [9].

The idea that domain-specific knowledge can reduce the search space is applicable to network search. For instance, when searching for information that involves a flow, the search process may progress along the path of the flow, as indicated in the next-hop field in the flow record. By doing so, we restrict the process from exhaustively searching the space to searching along a path.

The concepts proposed in this paper relate to the field of query processing in large database systems, which has been recently driven by technology companies that maintain large-scale ICT infrastructures, such as Google and Yahoo. These systems are characterized by a large number of servers that form the nodes of a distributed database, which stores logs from operational data, web content, etc. The relational data model has proved to be too rigid for these databases, and relaxed models, also called non-SQL databases, have been developed. Examples include Google's Dremel database system [33], together with the BigQuery query language [3]. Another example is Yahoo's Pig database system [23], together with the PigLatin query language [36]. Academic works include the ASTERIX system [14], a non-SQL database system, as well as the Weaver SQL system [28], which we developed in our lab.

As advocated by the above described research, we find that the relational model is too strict for our purpose and that the expressiveness of the relational algebra should be retained in the query language. The difference between a database system as described above and a network search system is their respective objective: a database system retrieves known data, while a network search system discovers potentially unknown data and relationships. Furthermore, in contrast to querying databases, ranking is important to network search, as well as imprecise results are useful, if they are obtained at low cost.

To realize continuous network search queries, the active research field of distributed stream processing is relevant. Queries in stream processing systems are executed on a network of processors, whereby sub-queries run in the processors, and data streams are pushed through the network. Examples of academic research in this field includes TAG [30], and Borealis [11], whereas IBM's System S [24] and Yahoo/Apache's S4 [35] are well known industrial efforts.

3. An Architecture for Network Search

Figure 1 shows an architecture for a network search system, which we introduced in [42]. Its key element is the *search plane*, which conceptualizes the network search functionality. This plane contains a network

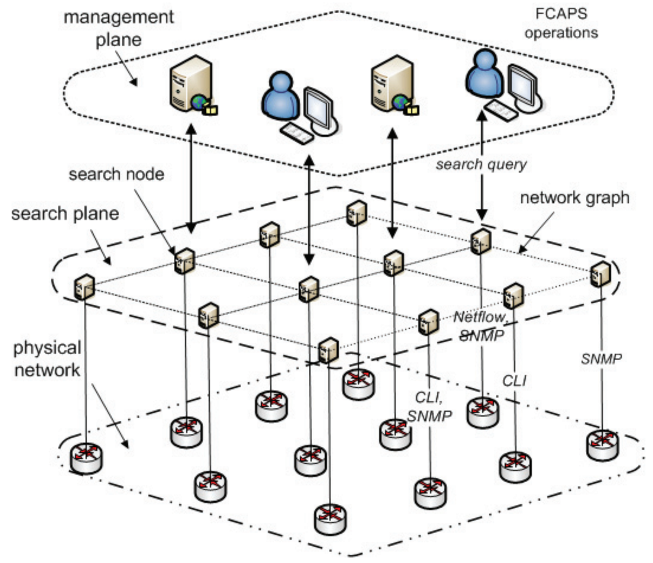

Fig. 1. An architecture for a network search system [42]

of *search nodes*, which have processing and storage capacities. A search node can communicate with a set of neighbors, which are identified through links of the network graph. The design of this plane supports searching in a distributed and parallel fashion. A search node can be realized in various ways: it can be part of the management infrastructure outside the managed system, it can be run as a standalone network appliance, or it can be integrated into a network element using a variety of technologies. Our current prototype implements the third option.

The bottom plane in Figure 1 represents the physical network that is subject to search. Each network element is associated with a search node, which maintains (or has access to) configuration and operational data from that network element. This data is modeled as a set of objects, whose structure is described in Section 4. Note that the figure shows the simplest form of association between a network element and a search node; it is possible that a search node maintains data from several physical devices, or, alternatively, a device updates data on several search nodes. The top of Figure 1 shows the management plane, which includes the systems and servers running processes for network supervision and management.

There are two important interfaces in this architecture. The first is the search interface, which supports the query language discussed in Section 5. We envision that every search node is an access point for search queries. The second interface defines the interaction between a search node and a network element, which can be realized through polling or can be push based. This interface is technology-dependent and possibly proprietary.

Each search node runs a process that communicates with the associated network element(s) from which it retrieves network data. A database function dynamically maps that data into the information model for network search and updates the local search database.

Search functions, invoked from the management plane through query invocation, are executed as distributed algorithms on the graph of search nodes. During the execution of a query on a search node, the local search database is accessed, the matching of the local query against stored indices is performed, and the local search result is possibly aggregated with results from other nodes.

4. Object Model

We consider physical and logical entities in a networked system, such as routers, servers, IP flows, virtual machines, etc., as *objects* in a *search space* (or *object space*). We associate an object in the space with each of these entities. An object is modeled as a bag of attribute-value pairs, containing configuration and operational information. An object is named and typed, and, hence, has at least two attribute-value pairs. Figure 2 shows two examples, an IP flow object with information available on a router, and a virtual machine object with data from a server.

We *name* an object using an Uniform Resource Name (URN) [34]. Such a name is a unique, location-independent and expressive identifier. (We choose URNs over Uniform Resource Locators (URLs) [15], because they are more persistent, and we do not consider oblivious identifiers, as used in [38], since they are not sufficiently expressive.) The top attribute-value pair in the objects shown in Figure 2 are URNs.

We introduce a relation between objects that links together objects that share attribute-value pairs. The relation will allow us to find information that belongs to a certain context in a networked system. For instance, it will enable us to trace an IP flow passing through the nodes of a network, or to search for the servers in a cluster that run applications belonging to a certain customer.

Consider objects a, b in a search space O. We say a is *directly linked* to b, denoted by $l(a,b)$, if a and b share an attribute-value pair. Obviously, $l(\cdot,\cdot)$ is reflexive and symmetric, but not transitive. Note that the same relation $l(\cdot,\cdot)$, with the same properties, can be defined on subsets $A, B \subset O$. We say $a \in O$ is *linked* to $b \in O$, denoted by $l^*(a,b)$, if

$$l^*(a,b) := \begin{cases} l(a,b), or \\ \exists c \in O : l^*(a,c) \wedge l^*(c,b) \end{cases}$$

The relation $l^*(\cdot,\cdot)$ is reflexive, symmetric and transitive, by construction, and, therefore, an equivalence relation. Similarly, as above, one can define $l^*(\cdot,\cdot)$ on subsets of O. It is often useful to compute the closure of

name	urn:ns:IPflow:128.146.222.233 :131.187.253.67:03FA:0016:06
type	IPflow
srcIPaddress	128.146.222.233
dstIPaddress	131.187.253.67
protocol	6
srcPort	03FA
dstPort	0016
packet	4
octet	1129
timestamp	10:05:11 24 April 2012

name	urn:ns:VM:instance-00000007
type	VM
uuid	4f5f86875be18e30c9000002
cpuCores	1
memory	1 GB
storage	3 GB
IPaddress	192.168.1.5
server	urn:ns:server:Server-08
customer	urn:ns:customer:John
dateCreated	12:49:25 10 February 2012

Fig. 2. Sample objects in the search space. On the left, an object representing an IP flow with information from a router; on the right, an object representing a virtual machine on a server.

a subset A under $l^*(\cdot,\cdot)$. For instance, all information associated with a video service can be found in the closure of the set of flow objects related to the service.

The above described model is simpler and coarser than the information models traditionally used in network management, such as, SMIv2 [32], GDMO [2], CIM [1], and YANG [16], but it is also less expressive. We believe that our model is better suited for network search, as one can formulate queries with minimal knowledge about information structure. Furthermore, one can easily populate our model with data from available sources in a network system, at the price of potentially losing structural information.

5. Query Language

A query on a of search space O returns a subset of information in this space. We describe the query language in BNF notation as follows:

$$\begin{array}{rrll}
\text{Basic:} & q \rightarrow & t \mid q \wedge q \mid q \vee q & (1)\\
& t \rightarrow & a \mid v \mid a \ op \ v & (2)\\
& op \rightarrow & = \mid < \mid \approx \mid \cdots & (3)\\
\text{Link:} & q \rightarrow & \lambda A \mid \lambda^* A & (4)\\
\text{Projection:} & p \rightarrow & q \mid \pi q & (5)\\
\text{Aggregation:} & r \rightarrow & p \mid \alpha p & (6)
\end{array}$$

First, we discuss queries q based on rules (1), (2), (3), which return a set of objects. Rule (2) defines how a query is made up of tokens t. a stands for an attribute name, such as *load*, v for a value, such as 0.7, and op for a relational operator. Here are some examples of queries based on rules (1), (2), (3): *load*, *load* $>$ 0.7, *server* $\wedge$ *load* $>$ *0.7*, *server* $\vee$ *router*, (server $\vee$ router) $\wedge$ *load=0.7*.

Each token t expresses a condition on an attribute-value pair. During query processing, the token is matched against all objects in the search space—more precisely, against all attribute-value pairs of objects in the search space. If the match is successful, then the object is included in the query result. For example, the token *server* returns all objects that contain the attribute name or value *server*. The token *load*>0.7 returns all objects that include an attribute named *load* with a value larger than 0.7.

Note that the match of a token to an attribute-value pair does not have to be exact, but can be approximate, for an object to be included in the query result. Approximate matching applies to value as well as to the attribute name. We consider approximate matching as an important characteristic of network search. The degree to which a token matches an attribute-value pair can be reflected in the ranking of the object in the query result. This issue is part of our future work.

The query $q_1 \vee q_2$ returns the union of the results of sub-queries q_1 and q_2. Likewise, $q_1 \wedge q_2$ returns the intersection of sub-queries q_1 and q_2. We give an example from a datacenter that offers Infrastructure-as-a-Service: "find servers with at least 12 CPU cores and that have load lower than 20 percent," which can be expressed as

$$\textit{server} \wedge \textit{cpuCores} > 11 \wedge \textit{load} < 0.2$$

A special case in matching occurs when the token contains a name. We allow a substring of the URN to be given as a name, for example, John instead of urn:ns:customer:John:Doe. If the substring matches the value of an object name, then the object is returned. This way, query processing performs name resolution.

Rule (4) describes *link* queries, whereby A denotes a set of objects, λ denotes the operator for direct linking, and λ^* denotes the operator for linking. In case of operator λ, the above query returns the directly linked objects of A. In case of λ^*, it returns the closure of A with respect to $l^*(\cdot,\cdot)$, which means all objects $o \in O$ that are linked to objects in A. The following query computes the closure of an object and returns objects of a specific type from that closure: "find servers that run processes of customer John".

$$\textit{server} \wedge \lambda^* \textit{ John}$$

Rule (5) introduces a projection operator $\pi_{a_1,\cdots a_n}$, which is applied to a set of objects and returns those attribute-value pairs whose attribute names $a_1 \cdots a_n$ are specified. This operator reduces the amount of information returned by a query, by giving back a subset of attributes for a particular set of objects, instead of all attributes of these objects. For instance, the query "find servers with at least 12 CPU cores" can be written as

$$\pi_{name}(\textit{server} \wedge \textit{cpuCores} > 12)$$

Rule (6) describes an aggregation operator $\alpha_{f,\,a}$, which takes as input a set of objects B and returns the value of the aggregation function f applied to the values of attribute a in all objects in B; i.e., $\alpha_{f,a}(B) = f(b_a | b \in B)$. Typical aggregation functions are *sum*, *count*, *max* and other statistical functions, although non-numerical functions can be used. This operator computes an aggregated value of a set of objects, rather than returning the objects themselves. The query "find the virtual machine with the highest CPU cores on server X" can be expressed as

$$\alpha_{max,\,cpuCores}\ (\textit{VM} \wedge \textit{*server*X})$$

We briefly compare our object model and query language with that of the relational model with its standard operators selection, projection, aggregation and semi-join [20]. If we restrict our object model in such a way that objects of the same type have a predefined set of attributes only, and tokens are of the form $a\ op\ v$ only, then *our model becomes as expressive as the relational model*, in the following sense. An object in our model corresponds to a tuple in the relational model and vice-versa; objects of the same type correspond to a relation and vice-versa. Now, it is straightforward to show that queries on our model produce results that corresponds to those results the operators on the relational model produce. For example, a basic query made up of rules (1), (2), (3) corresponds to a selection operation (possibly with set union or intersection), a (direct) link query expressed by rule (6) corresponds to a semi-join operation, etc. To see the latter, consider two set of objects A_1 and A_2 and their relations R_1 and R_2. It is then straightforward to see that the query $A_1 \wedge (\lambda A_2)$ in our model produces the result that corresponds to the relational query $R_1 \ltimes R_2$.

While our model is as expressive as the relational model, it is more general. First, it allows for objects of the same type to have different attributes. This generalization allows us to capture the heterogeneity of network entities. For instance, network functions like firewalls come in different varieties and, therefore, need to be described using different attribute sets. Second, our model replaces the tuple identifier in the relational model, i.e., the key attributes, with a single attribute, namely, the object name. Third, in our model, an attribute name or an attribute value, can be given as a query, which is not possible in the relational model. This allows us to invoke queries without explicit knowledge of the object structures, and it enables us to give values without providing the corresponding attribute names as well. Lastly, the link operator in our model does not the have a corresponding operator in the relational model (although the direct link operator has, as discussed above).

6. Distributed Query Processing

Our approach to process network queries makes use of the echo protocol, a tree-based protocol suitable for distributed polling [40] [29]. It is based on an algorithm first described by Segall [37]. The execution of the echo protocol can be understood as the subsequent expansion and contraction of a wave on a network graph. The execution starts and terminates on an initiating node of the graph, also called the root (node). The wave expands through explorer messages, which nodes forward to their respective neighbors. During the expansion phase, local operations are triggered on the nodes after receiving an explorer. The results of these local operations are collected in echo messages, when the wave contracts, so that the aggregated result of the global operation becomes available at the root node. During the expansion phase, the protocol constructs a spanning tree on the network graph for the purpose of collecting and aggregating the partial results during the contraction phase.

The echo protocol executes on the network graph of the search plane (Figure 1). The protocol can be started on any search node once a query q has been received. First, the query is disseminated by explorer messages to every node and executed as local operation against the local database D. The results of the local operations are sent, by echo messages, on the spanning tree from child nodes to parent nodes, where the partial results are aggregated. (Note that the term aggregation here refers to the processing of the partial query results, not to a possible aggregation operator in the query q.) Figure 3 shows a sample spanning tree created by the echo protocol on nodes $n_1, \cdots, n_6$ with root n_1. It further shows the message exchange between nodes. Each node shows the local database D containing the objects with information from that node, together with the variable *result*, which contains the (partial) result of the query q.

The definition of the local operation, the aggregation operation of the query result, and the current local state of the query collection, are modeled in an object, called the *aggregator object* of the echo protocol (see Figure 4). The aggregator in the figure contains the code to process a query q that contains neither λ nor λ^* operator. Line 2 defines the variable $result$, which is either a set of objects or, it contains an aggregated value of this information in case the query contains an aggregation function. Lines 3-4 defines the local function and lines 5-9 defines the procedure how partial results are aggregated. If the query q includes an aggregation operator $\alpha_{f,a}$, then variable *result* is updated using the aggregation function f_a, otherwise by computing the union of the partial results.

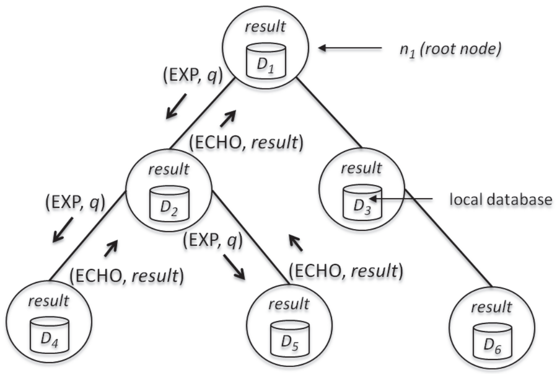

Fig. 3. Distributed query processing: The echo protocol creates a spanning tree in the search plane. Each node contains the local database D. The variable $result$ contains the partial result of a query q. Some of the explorer (EXP) and echo (ECHO) messages are shown.

```
1: aggregator object processQuery( )
2:     var: $result$:QueryResult;
3:     procedure local( )
4:         $result := q(D)$;
5:     procedure aggregate($childResult$:QueryResult)
6:         if $q$ contains $\alpha_{f,a}$ then
7:             $result := f_a(result, childResult)$;
8:         else
9:             $result := result \cup childResult$;
```

Fig. 4. Aggregator for processing a query without link operators.

We describe now the processing of queries q that contain link operators. First, if q is of the form λA, whereby A is a set of objects, then the local function in the aggregator object can be written as $A \cup \{o \in D - A \mid \exists a \in A : l(a, o)\}$. Second, if q is of the form $\lambda q'$, then processing q requires two executions of echo; during the first execution, q' is computed, and the second execution computes the directly linked objects. Finally, in case q is of the form $\lambda^* q'$, at least two executions of echo are required, whereby, first, q is processed, then $\lambda q'$, $\lambda(\lambda q')$, $\lambda(\lambda(\lambda q'))$, etc. The process stops when the output set does not grow from one execution to the next.

The proof that the above described distributed processing technique is correct draws from the fact that the global database, which contains all objects, is horizontally fragmented; each object is stored in exactly one local database. The performance characteristics of distributed query processing is based on the performance properties of the echo protocol [40]. For instance, the execution time of a query grows proportionally with the height of the spanning tree, which is upper bounded by the diameter of the network graph. The protocol overhead is evenly distributed on the network graph, as two messages traverse each link during the execution of echo. Lastly, the number of messages each search node processes is upper bounded by the degree of the network graph. While the above properties suggests that the presented generic approach to network search is a scalable solution, major challenges remain to engineer a large-scale network search system. See Section 8.

7. Use Cases

A. Use Case: Networking

The delivery of complex networking services involves increasing numbers of entities with numerous interdependencies, many of which are only temporary and fleeting in nature. With the advent of cloud services and virtualization, many of the entities themselves are increasingly short-lived, as layers of virtualized entities build on each other. In addition, many decisions that affect end-to-end service characteristics depend on very dynamic decisions, such as performance routing in which the path of individual packets in a media stream depends on short-lived fluctuations in load and service levels. As a result, management paradigms need to increasingly shift away from planning and predicting, towards monitoring and tracing (packets in a network, dependencies between resources, policy decisions imposed on a flow, etc.), in order to understand what is precisely going on, should it be required.

However, tracing, while tremendously important, can be a challenge. This is where network search can provide some distinct advantages. Consider the case in which an operator is concerned about an interactive video session whose service level is deteriorating. The operator has no way of knowing which particular systems or links the stream is traversing and might contribute to the problem. Using network search, the operator can issue a network query to retrieve a list of all systems in the network with a flow record which contain the source and destination systems as part of their flow keys, along with the flow records and the interface statistics of the incoming and outgoing interfaces. The search provides the operator with data from every node in the network that participates in the flow. From this, an application can perform further analysis and stitch together which path the flow takes, whether multiple routes (perhaps due to load balancing considerations) are taken, where packets are dropped, etc.

Now consider the same scenario but involving networking boundaries at which NAT (Network Address Translation) is performed. For example, the session might involve two endpoints in two different enterprises and traverse a public service provider network. In that case, network address translation occurs at the border between enterprise A and the service provider, and again at the border between the service provider and enterprise B. As a result, the same session is associated with different flow keys in each of these domains. Hence, network search cannot be applied naively in the same query being issued to every device in either of the involved networks. Instead, the query itself needs to be locally adapted according to the network address translations at the different network boundaries—the network search needs to be NAT-aware. For this purpose, we introduce NAT-aware search nodes at nodes that involve a NAT function. A NAT-aware search node has access to the same NAT database as the NAT function itself. Whenever a search query or a search result involves an IP address in the NAT database, the NAT transformation is applied as the query traverses the network boundary.

B. Search on a Cloud Infrastructure

In order to experiment with network search concepts, we have instrumented a cloud Infrastructure-as-a-Service (IaaS) platform in our laboratory with network search functions. The platform contains nine high-performance servers, interconnected by Gigabit Ethernet, and runs the OpenStack cloud management software. (See [44] for details.) The components of the network search system include search nodes—each server on the testbed runs such a node—and managers in the management plane that run in a management station. Each search node contains a local database based on MongoDB [5] that maintains the network objects. Currently the database has four types of objects, namely, server, virtual machine (VM), application, and customer. A data sensing component reads system files, such as '/proc' [6], libvirt configuration [4] etc., and populates and periodically updates the objects in the local database at a rate corresponding to their respective lifetime. A distributed query processing component implements the protocol outlined in Section 6. The manager component offers two types of interfaces for accessing network search functionality: a simple line console and a graphical, menu supported interface that allows to compose queries and browse the output in various ways. Due to lack of space, details about this implementation will be reported elsewhere.

We have used the network search system on the OpenStack platform for conducting a range of exploratory experiments. The platform can be loaded by external load generators that were developed for evaluating performance management solutions for OpenStack [44]. The produced load has a time-varying pattern of several types of applications running in virtual machines of different configurations and lifetimes. Here are some of the experiments we have performed. First, we inquire about the load on a server cluster, which is given by a range of IP address: $\alpha_{sum,\,load}$ *(192.168.212.*)*. Given the case that the load is unexpectedly high, we want to find out which applications are running on this cluster: $\pi_{name}($ *application* $\wedge$ λ^* 192.168.212.*). Finally, we want to identify the customers for which these applications are executed: $\pi_{name}($ *customer* $\wedge\ \lambda\ app_1 \vee \cdots \vee app_n)$. In a further set of experiments, we study the behavior of the virtual machines on the platform under an adaptive placement policy. First, we are interested in learning the distribution of the uptimes of the virtual machines. The query $\pi_{uptime}(VM)$ provides the uptime of the active virtual machines, out of which the distribution can be computed. (If a distribution aggregator is implemented, then the distribution can be directly computed as part of the query). Second, we want to study the movement of virtual machines that belongs to a specific application. We can achieve this by periodically issuing the query $\pi_{name,\,server}$ $(VM \wedge app_x)$.

We ran a series of performance tests on the search system, the details of which can be found in [39]. For some experiments, the search space was populated with 2000-3000 objects on each server, which took up some 1-1.5 MB of disk space. In the current configuration, objects are refreshed once per second via polling, which creates a CPU load of about 2 percent on each server, for local databases with 2000 objects. Global search queries, which include projection and aggregation operators, execute in approximately 20 milliseconds, while link queries can take considerably more time. An analysis of the performance measurements suggests that there is a room for performance improvements in the implementation of the current prototype.

8. Discussion

The contribution of this paper centers around a simple query language for search in networked systems. Queries are based on a model where objects are represented as a set of attribute-value pairs. We propose a method for distributed execution of search queries in a networked system in which nodes maintain objects that contain configuration and operational information. We argue why the proposed method provides the correct result for a network query. Two use cases further motivate the paradigm of network search and demonstrate that the introduced language is useful. Our implementation gives evidence that the design can be implemented (even if the testbed is of limited size).

Similar to the case of web search, the simplicity of our query language has the drawback that, often, the information we are interested in cannot be expressed in a sufficiently precise manner, and, therefore, the query result needs interpretation. For instance, the query "search for servers that run processes of customer John" cannot be directly expressed in our query language, but only through attribute names, values, and object links, for example, through *server* $\wedge$ λ^* *name = *John*. This query returns objects with attribute name (or value) *server* that are linked to objects whose names end with *John*. The query result needs to be interpreted and, hopefully, contains the information we searched for in the first place.

An argument can be made that search queries can be implemented as specialized protocols in a networked system, and, therefore, a generic search system is not needed (see Section 7-B). However, we believe that a network search system enables new functionality to be added on-demand, or it allows for network applications to dynamically adapt their information demand.

In future work, we plan to further develop the paradigm of network search. Here are some of our priorities. While the contribution in this paper focuses on database aspects of network search, network search includes concepts that go beyond database functionality, most importantly, approximate matching of attributes and ranking of search results (see Section 1). We envision, for instance, that ranking takes into account the freshness of the data, the locality of the query invocation, and the number of tokens in a query that matches a particular object, and that the ranking process is realized as a distributed aggregation function.

Second, we plan on improving the scalability of distributed query processing for network search. While this paper describes a distributed method for query processing, each query still invokes an operation on every search node, which is expensive in a large system. We are considering several approaches to reduce the footprint of a query. As pointed out in Section 2, domain knowledge can be used to guide the search process and thus reduce the search space. Alternatively, an index structure can be developed to reduce the number of nodes that are involved in processing a query. Link queries require special attention, since each such query involves several executions of echo. Possible heuristics for reducing the overhead include restricting the search to those links with the number of intermediate objects below a given bound, and limiting the subsequent executions of echo to those nodes that produced a non-empty query result during the previous execution.

Additionally, work is needed for the development of efficient local databases, the population of local database with available data sources, the development of concepts regarding the privacy and security of local data, as well as a framework for search in a multi-domain environment.

REFERENCES

[1] Common information model. “http://dmtf.org/standards/ci”, August 2012.

[2] GDMO - Guidelines for Definition of Managed Objects. “http://www.cellsoft.de/telecom/gdmo.htm”, August 2012.

[3] Google BigQuery. “https://developers.google.com/bigquery/docs/overview”, August 2012.

[4] libvirt 0.7.5 - Application Development Guide. “http://libvirt.org/guide/html/”, August 2012.

[5] MongoDB Manual. “http://docs.mongodb.org/manual/”, August 2012.

[6] proc. “http://manpages.courier-mta.org/htmlman5/proc.5.html”, August 2012.

[7] RSS 2.0 specification. “http://www.rssboard.org/rss-specification”, August 2012.

[8] Wikipedia. http://www.wikipedia.org/”, August 2012.

[9] Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Cong Yu. Socialscope: Enabling information discovery on social content sites. In *CIDR*, 2009.

[10] Mike Andrews. Searching the internet. *IEEE Software*, 29:13–16, 2012.

[11] Magdalena Balazinska, Hari Balakrishnan, Samuel R. Madden, and Michael Stonebraker. Fault-tolerance in the borealis distributed stream processing system. *ACM Trans. Database Syst.*, 33(1):3:1–3:44, March 2008.

[12] Nilesh Bansal and Nick Koudas. Searching the blogosphere. In *WebDB*, 2007.

[13] Shenghua Bao, Guirong Xue, Xiaoyuan Wu, Yong Yu, Ben Fei, and Zhong Su. Optimizing web search using social annotations. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 501–510, New York, NY, USA, 2007. ACM.

[14] Alexander Behm, VinayakR. Borkar, MichaelJ. Carey, Raman Grover, Chen Li, Nicola Onose, Rares Vernica, Alin Deutsch, Yannis Papakonstantinou, and VassilisJ. Tsotras. Asterix: towards a scalable, semistructured data platform for evolving-world models. *Distributed and Parallel Databases*, 29:185–216, 2011.

[15] T. Berners-Lee, L. Masinter, and McCahill M. Uniform resource locators (URL). IETF, RFC 1738, December 1994. May 2012. ¡http://www.ietf.org/rfc/rfc1738.txt¿.

[16] M. Bjorklund. RFC 6020: YANG - A Data Modeling Language for the Network Configuration Protocol, October 2010. May 2012. <https://tools.ietf.org/html/rfc6020>.

[17] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[18] Chun Chen, Feng Li, Beng Chin Ooi, and Sai Wu. Ti: an efficient indexing mechanism for real-time search on tweets. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 649–660, New York, NY, USA, 2011. ACM.

[19] Abhishek Das and Ankit Jain. *Indexing the World Wide Web: The Journey So Far*. 2011.

[20] C.J. Date. *Introduction to Database Systems*. Addison-Wesley, 8 edition, 2003.

[21] Anlei Dong, Yi Chang, Zhaohui Zheng, Gilad Mishne, Jing Bai, Ruiqiang Zhang, Karolina Buchner, Ciya Liao, and Fernando Diaz. Towards recency ranking in web search. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 11–20, New York, NY, USA, 2010. ACM.

[22] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets*, chapter 14, pages 397–435. Cambridge University Press, 2010.

[23] Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a high-level dataflow system on top of map-reduce: the pig experience. *Proc. VLDB Endow.*, 2(2):1414–1425, August 2009.

[24] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. Spade: the system s declarative stream processing engine. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1123–1134, New York, NY, USA, 2008. ACM.

[25] G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 953 –962, april 2008.

[26] Anne-Marie Kermarrec. Challenges in personalizing and decentralizing the web: An overview of gossple. In Rachid Guerraoui and Franck Petit, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 5873 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2009.

[27] Xiaonan Li, Chengkai Li, and Cong Yu. Entity-relationship queries over wikipedia. In *Proceedings of the 2nd international workshop on Search and mining user-generated contents*, SMUC '10, pages 21–28, New York, NY, USA, 2010. ACM.

[28] Koon-Seng Lim and Rolf Stadler. Real-time views of network traffic using decentralized management. In *Integrated Network Management*, pages 119–132, 2005.

[29] K.S. Lim and R. Stadler. A navigation pattern for scalable internet management. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 405–420. Ieee, 2001.

[30] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, December 2002.

[31] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[32] K. McCloghrie, D. Perkins, and J. Schoenwaelder. RFC 2578: Structure of Management Information Version 2, April 1999. May 2012. <http://tools.ietf.org/html/rfc2578>.

[33] Sergey Melnik, Andrey Gubarev, Jing J. Long, Geoffrey Romer, Shiva Shivakomar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of Web-Scale datasets. In *The 36th International Conference on Very Large Data Bases*, volume 3, September 2010.

[34] R. Moats. RFC 2141: URN Syntax, May 1997. May 2012. <http://datatracker.ietf.org/doc/rfc2141/ >.

[35] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *ICDM Workshops*, pages 170–177, 2010.

[36] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[37] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, 29:23–35, 1983.

[38] A. Shieh, E. G. Sirer, and F. B. Schneider. NetQuery: a knowledge plane for reasoning about network properties. In *Proceedings of the ACM CoNEXT Student Workshop*, CoNEXT '10 Student Workshop, pages 23:1–23:2. ACM, 2010.

[39] Amy Skinner. A system for googling operational data in clouds. Master's thesis, KTH The Royal Institute of Technology, December 2012.

[40] Rolf Stadler. Protocols for distributed management. Technical Report 2012:028, KTH, Communication Networks, 2012. QC 20120604.

[41] Jaime Teevan, Daniel Ramage, and Merredith Ringel Morris. #TwitterSearch: a comparison of microblog search and web search. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 35–44, New York, NY, USA, 2011. ACM.

[42] Misbah Uddin, Rolf Stadler, and Alexander Clemm. Management by network search. In *NOMS*, pages 146–154, 2012.

[43] Monique V. Vieira, Bruno M. Fonseca, Rodrigo Damazio, Paulo B. Golgher, Davi de Castro Reis, and Berthier Ribeiro-Neto. Efficient search ranking in social networks. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 563–572, New York, NY, USA, 2007. ACM.

[44] Fetahi Wuhib, Rolf Stadler, and Hans Lindgren. Dynamic resource allocation with management objectives : Implementation for an openstack cloud. Technical Report 2012:021, KTH, Communication Networks, 2012. QC 20120528.

[45] Sihem Amer Yahia, Michael Benedikt, and Philip Bohannon. Challenges in searching online communities. *IEEE Data Eng. Bull*, 30:2007.