Data-Aware Networking: Routing queries in a federated environment

Jordan Augé Marc-Olivier Buob Loïc Baron Timur Friedman Serge Fdida

1. INTRODUCTION

The amount of data that is produced today in various sectors, including the scientific community, is growing at an unprecedented rate. Numerous applications require that data belonging to independent organization can be distributed over the network to be processed, combined and analyzed. The measurement and monitoring of Internet paths, crossing multiple domains, is an instance of such an issue, requiring cooperation by several independent entities. Much research so far has consisted in proposing new architecture and protocols designed for specific federations.

In this paper, we propose a new direction for handling data flows on the network, which draws upon the knowledge – both theoretical and practical – we have acquired on the Internet, and borrows at the same time from fields such as databases. We propose some foundations for a decentralized architecture, allowing independent actors to exhange data and form an ecosystem built around the exchange of value. Although an ecosystem of potential entities already exists today, it is fair to say that no infrastructure allows its materialization yet.

This is the intent of this paper to propose an infrastructure allowing to put data as a first-class citizen of networks. We name it *Data-Aware Networks*, in the sense that the network will be able to route the queries to the appropriate hosts able to answer it, and support the transit of the data back to its requester. Just like packets are the basic unit considered on IP networks, grouped consistenly within flows, we propose to consider the transport of data units triggered by a query, eventually involving the cooperation of several independent, federated entities.

The foundations of such Data-Aware Networks are presented in Section **??**, and the requirements for a communication protocol allowing the transport of queries between the different entities is proposed in Section **??**. We then consider how queries are handled within a single domain in Section 4, and the routing mechanisms we propose in Section 5. We then illustrate our proposal through some sample applications, and present MANIFOLD, an implementation of the concept that is powering two applications, namely TopHat [**?**] and MySlice [**?**]. Finally, Section **??** presents some related work.

Shall we announce the advantage of such an architecture already here ?

2. FOUNDATIONS OF DATA-AWARE NETWORKING

2.1 The ecosystem of data

When we said previously that there is not a materialized ecosystem for data exchange in the network, it is more correct to notice that there is in fact a multitude of sparse efforts, each focused on a particular application. Let's look at Internet measurements, as an illustration. So far, there exist many independent efforts to measure its many facets, and more and more the need arise to be able to get access to all those different sources of data and combine them to extend our comprehension of the way it works. For various reasons, it would be impossible for a single entity to have the necessary knowledge, or to afford to build a global and versatily measurement platform. Instead, we see many efforts to propose datasets and real time data streams, in various formats, that can be used as building blocks for creating more advanced systems.

The collaboration between those entities relies mainly on their mutual interest, which mean that both find some value in mutualizing those resources (complementarity, scalability, sustainability, scale economies, etc.). **We could refer to a paper about federation...** Though, this collaboration is hindered by a vast set of technical barriers preventing an easy interconnection: data is often not consistenly stored or retrieved, it is named differently, lacks some units, etc. Successful integration thus relies on a large amount of technical work, which is also what prevents scientific datasets to be easily and openly shared. We believe that those tasks, while not being fully transparent, should benefit from an agreed infrastructure and some support from the network.

It is of course a pipe dream to expect that it is possible that the network implement specific protocols for each type of data to be transported, or every way to contribute value to the ecosystem. We can think of entities providing measurements, caching, large storage, computing resources, aggregation and translation services, etc. But there are thousands more, and nothing should prevent a new entity to emerge if tomorrow it is able to bring some value for at least one member of the community[1]. We could make the analogy with the IP networks whose openness have allowed Content Providers to emerge and find they way in the ecosystem.

It seems natural to consider to what extent such an architecture can be extended and adapted to be applied to data, since its prevalence today deserves some additional support. We are thus considering a fully decentralized architecture, relying on a set of independent domains contributing value, which are free to negociate and set up peering agreement

[1]This could be materialized by an exchange of money for example

between themselves.

2.2 The status today

In order to understand the choices that lays the grounds for our work, we need to have a look on the status of data today, and the fundamental requirement it has to support. Let's start with a few observations.

Data is highly heterogeneous.

- databases, flat files, XML, webservices,
- many concurrent protocols and data formats
- units, sampling,
- temporal aspects, annotations
- data flows: archive vs streaming vs on demand measurements

Data is scattered across numerous entities.

- aggregation that make some ground for routing
- ecosystem will also naturally evolve (economic incentive).

Many unsuspected data are complementary.

The ecosystem is huge and highly dynamic.

2.3 Overview of Data-Aware Networks

The architecture we propose aims at tackling those different challenges. First, we propose and motivate the need for a standard way to represent and transport data between end hosts, able to cope with a vast diversity of applications. Then, because for various reasons, the data will be split and distributed across several domains, and several hosts within a domain, we propose an adapted query mechanisms, only relying on a high level description of the data. **This place the data at the center of the network, not its location or its ownership.** A routing layer will be in charge of mapping such queries with the different entities able to answer it, both within a single domain, or across the federation. We note though that it is possible to explicitely target or exclude some known platforms during the query. **Routing = added value.**

Minimal amount of information to be shared to allow for a global federation

We will insert a schema illustrating the different domains: intra- and inter-domain, policies and agreements, specialized link or common substrate

Here we address dynamic and scalability concerns. size of routing tables parallel overlays = interests/community only information at the edge of the network (see later redundant section).

How to we adress multiple hops ?

Economical incentive, model, value sharing, etc discussed in Section 9.

2.4 Representation of data

Relation model.

We adopt the format of the relational model, which represent data as a multidimensional tuple associating a value to a set of attributes. We group the set of tuples having the same attributes within a class, which is identified by a key (a subset of attributes uniquely identifying the tuple within the class). The attributes can be used to reference an other class instance through their primary key as usual. The major difference with the relational model is that we allows those objects to possess methods. Such a representation has proved to be general enough to accommodate the different types of data that will be represented.

A particularity of our model is that the objects are not defined by a single platform, but instead are created though the joint declarations of the various platforms. Thus the different properties of an object might be spread over several platforms. Also, an object can be hosted on several platforms at the same time (example of caches), through for the sake of simplicity, we will assume here that there are no contradictions.

Use of ontologies.

The framework we are proposing in this article is agnostic of the different information it transports. Thus since we want to be able to unify and join information originating from various sources, we somehow need the different entities to use the same naming scheme for similar objects. We decided to rely on ontologies, which formally describe some agreed and simplified representation of a domain of knowledge. For example, we present some efforts in expressing ontologies for measurements in Section **??**.

Such ontologies fit our need as they define a semantic graph involving objects together with their attributes, their relations and their inheritance properties. For the sake of simplicity, we will limit ourselves to the case where all exchanged information belongs to a single domain, and that all platforms agree on the same ontology. In practise, we might extend our model to support multiple ontologies, and we could for instance consider new actors in charge of translating or bridging the information originating for several domains.

In practice, such graphs are huge and might hold several thousands of concepts and attributes. It is both unrealistic to assume that all platforms will expose the full set of attributes, and would pose serious scalability concerns for the elements of our network if they were to be aware of the full graph. The platform will typically only expose a small fraction of these objects and attributes (like a view in a database). Assuming the platforms agree on the underlying semantic, we will see that this information will be sufficient so that the intermediate nodes are able to reconstruct the subset of interest from the semantic (see Section 5.

3. COMMUNICATION PROTOCOL

We now assume that all platforms uniformly represent their data in the scheme describe above. We now need to define a way for a user to communicate and express a query on the data of interest for him.

3.1 Data manipulation language

The interaction with the different platforms will be performed by contacting a reference platform (the first hop) that will eventually be in charge of relaying parts of the request to third-party federated platforms. We denote this request as a **query**. Queries are intended not only to get able to retrieve data or trigger new measurements, but also to be able to manipulate the set of existing data on the platform (creating, deleting or updating existing elements). These operations are really similar to those one could perform on a database, expect we are communicating with the federation at large: `Get`, `Create`, `Update`, `Delete`[2]. As objects in our framework possess callable methods, we extend our data manipulation language with an `Execute` keyword.

3.2 Addressing data

In a database, queries contain a precise description of the information of interest in the form of both attributes and the table they belong to. The heterogeneity of the different platforms, which can store the data differently from one to another, requires us to relax those constraints, so that all data can appear like originating from one single domain. That means someone issueing a query should not be required to know which domain is responsible for it (the partitioning of information) and how it is stored. As we will see in further sections, the network will expose a schema of available information to its users (composed of a set of views) on which it can express its queries consistently. We note that this requirement does not prevent the annotation of data with its provenance for transparency issues.

It is not realistic to assume that we can assign addresses or identifiers to each data unit, and effectively use them for routing matching the user query with the related tuples. This would cause major overhead cause of the lack of possible way of aggregating the description of data available on a platform; this would also require some global synchronization in the case of duplicated data (like in caches). Instead, we will rely on addressing the data by their class, together with the set of attributes of interest. Together with the description provided by the platforms, this information will be sufficient to deduce the set of tables that need to be combined to reconstitue partitions and views. This is what we refer by routing our query. It can be either done within a single domain, or elicit some external queries to other platforms to cope with locally missing information.

At this step, we can make an analogy with data warehousing solutions, that typically allow users to retrieve data though a set of dimensions that enhance the data available for a given fact (the class in our case). Those attribute can eventually be hierarchically organized. This combination of fact and dimensions is often refered to as a star, or snowflake schema.

3.3 Restricting the scope of queries

The set of attributes for an object will result from the merge of local and remote information (we expose a view), and the range of their value will typically be covered by several platforms (we group partitions). In order to restrict the scope of a query, a user will be invited to provide within it query a set of fields, filters and when necessary some parameters:

- Fields allow a user to specify which attributes from an object he wants to see returned: the result of a query will consist in the set of tuples matched by the query, having only requested attributes (whose value can in turn be a set of tuples in case of a nested query).

- Filters allow to restrict the range of a query on some object though a set of predicate of the object attributes, triples in the form (attribute, operator, value): the query will thus only match a partition of the available data.

- Finally, update queries allow to set the value of some attributes, which will have to be transmitted within the query as parameters.

[2]often denoted CRUD, which stands for Create/Read/Update/Delete, or INSERT/SELECT/UPDATE/DELETE in SQL, or POST/GET/PUT-PATCH/DELETE in REST.

Those operations will require contacting the different platforms responsible for the aggregate set of attributes involved for the query. Routing will be performed on this set relatively to the object class.

R/W is also necessary for routing both on table and on attributes. We might skip such details in the article.

Other operators are considered such as SORT, LIMIT and OFFSET that will make a consistent navigation possible in the collection, as well as offer the base for more efficient processing. Such operators as well as other extensions will be proposed in future work, and won't be further described in this document.

3.4 Communication between peers

Let's now consider two hosts, identified by their IP address, one issueing a query for the second one. We can consider two communication schemes, that we will denote as *pull* and *pull*. They form the base of our communication architecture, that will allow the implementation of several behaviours, as we will see in Section 6.

Pull scheme.

In the *pull* scheme, the answer to a query will be addressed back to the emitter through in the reverse direction of the query. That means the data will flow backwards in the query plane (a tree) up to its root, which is connected to the previous hop. This is either the emitter of the query in the single domain case, or a previous domain which routed the query until this point. That means that a router will have to maintain an active query table, so that it can map incoming answers to their source peer. **Analogy with flow tables in flow-aware routers, cf FAN, Caspian, Anagran.**

This behaviour is adapted for one-time queries, and follows the expected behaviour of the answer flowing back to the source in the reverse direction.

Push scheme.

Sometimes, we might want to consider asynchronous schemes. This is useful for supporting alerts, or callback, asynchronous or periodic queries. It can also be used to address the result to a third party.

This is also convenient when the routing is just forwarding the data several times without modifying it. The source then has better directly addressing the data to the original host to avoid wasting network resources.

In practice, data might still have to be pushed to one or many intermediary hops since they might be involved in some processing. This behaviour will be resolved during the query plane establishment. We note it will also require some information about the support of such functionality in the different platforms.

Do we want to push further the IP analogy and present a possible data format for a communication protocol built on top of IP ?

4. INTRADOMAIN ROUTING: THE QUERY PLANE SEEN AS A ROUTE

In this section, we suppose we have a single entity providing data. For the sake of convenience and analogy, we will assume it consists in a SQL database, and we will voluntary restrict ourselves to retrieving data. We also suppose the range of possible queries is known by the user; it corresponds to the set of possible views over the database schema. **We need to define which views we are considering : LEFT JOINS.**

Answering a query will be done in two distinct phases: **query routing** and **query optimization**, that respectively consist in choosing a set of candidate routes, or query planes, and selecting the cheapest one.

4.1 Query routing

The major difference between our query format and SQL is that we do not specify in which table the different attributes can be found. Instead, we rely on our unique naming assumption, that will allow us to find the table that need to be joined to answer the query. We will denote this step as **query routing**, and we will also consider the selection of necessary partitions.

Determine intermediary tables.

Given a set of requested fields, we will have: Route(fields) = JOIN(UNION(Tables))

4.2 Query optimization

Overview.

This specification in fact regroups a range of various query planes, whose efficiency will depend on the join order for example. Further more, when applying the different operators (selection, projection, etc) for processing the data, many query planes with various costs will be generated, and it will be the role of the **query optimization** engine to choose the cheapest one.

Cost model.

The notion of cost is voluntarily vague since it will depend on the user choice and the range of available information: order of operation and their cost (memory, cpu, storage, network), number of tuples to be materialized, etc. We refer the reader to the relevant litterature from the database community for an overview of the different techniques that are commonly used.

Simple heuristics.

Join order : left tree, and not bushy (no parallelism considered at the moment). Some joins need to be done before others. eventually, the one involving as few tuples as possible ?

Then we will consider a series of equivalent query plans applying algebraic rules. Heuristic that consist in replacing the current query plan with an equivalent one with a cost supposed inferior.

4.3 Dataflows

Database network convergence. in the context of adaptive dataflows. query processing = adaptive dataflow

[Hellerstein: Cosmic convergence] implementation : pipelining operators. subexpression materialized. implementation more flexible than algebra. impose a router.

query plane will consist in a data flow.

4.4 Database normalization

In this section we describe the interest of database normalization for queries. **Later in routing, we will discuss about the interest of the normal form for routing : fewer changes, formula, etc.**

5. ROUTING DATA QUERIES

5.1 Peering

So far, we presented how a single entity can answer a query. In order to fully exploit the value offered by the federation, it is desirable that a platform is able to relay a query to another one, or even relying on the combined information from several platforms to answer its query. Like for the interdomain agreements on the internet, we will suppose that the different domains will negociate peering agreements, allowing them to interconnect and establish sessions with other peers.

The different domains will remain autonomous in the sense that they can use the technology of their choice for the routing of queries within a domain (cf the example of the SQL database in the previous section); they just need to expose standard interfaces to the federation, like it is the case in the internet with the BGP protocol.

In this section, we complement the communication and transport protocol we have presented so far by an announcement protocol, and a routing process. The former is used by the different platforms to disseminate information about the routes toward the data, in other terms the subset of the global schema that is accessible through this peer. The latter is used to efficiently answer the incoming query by choosing the best route, in term of a cost metric that is to be defined.

We expect this proved scheme to be adapted to our current purposes, since it allows the different platforms to keep their entire autonomy. Like in the internet, they are free to implement outbound and inbound policies that will select which information they are keen on sharing and receiving from the peers.

SCHEMA HERE

Do we need to say more about policies ?

5.2 Announcements

Structure of announcements.

Once a platform has negociated a peering agreement with another one, it can use it to relay queries, or part of queries, it cannot answer by itself only. This requires them to be able to exchange metadata information about queries they are able to answer, and keep this synchronized. This means transfering the local view of the global schema that the platform has, consisting of objects with attributes and methods, eventually annotated with some partitioning information. We will denote this a route, by analogy with BGP announcements.

As we have seen, a query might also contain some processing instructions, like the choice of returned attributes, some filters, etc. Because of the heterogeneity of platforms, not all of them cant natively support such operations. We further allow platforms to expose the set of operations they support, from a base set containing in fact the operators from the relational algebra on which we build. For example a SQL database might advertise it support of column selection (named projection, the SELECT clause), or filtering (selection, the WHERE clause), etc. while flat files might just inform they only allow retrieving the full set of data. In such cases, the platform issueing the request will have to perform those operations by itself. The advantage of such a framework is that the number of useful operators to be supported is limited to a dozen or so.

Finally, those route will transport some cost information, or metrics that will make it possible to choose the best route. We leave this aspect open for further work, and note that depending on the algorithm we will use for computing routing information, the choice of this cost might determine the quality and the convergence of the routing **Need review from Marco here.** We refer the reader to [?] and further references, for such considerations applied to interdomain routing in the Internet.

Announcement = (Route, Capabilities, Cost metrics)

Normalization.

Because different peers might give access to the same, to complementary or even overlapping data, the announces received by a platform might not be fully consistent. Under our assumption of a shared semantic, this is as if they were all announcing a set of views on top of the same underlying schema which is the semantic graph we introduced above.

Thanks to database normalization technique (and we are particularly interested in the third normal form), it is possible to reconstruct from the different announces a refined schema reconciling the different platforms. Of course because the lack of information, this schema will still be far from the real underlying graph, but it will be sufficient considering the small subset of it which is exposed through the federation.

The renormalized schema will form the base information that will be used for routing queries to the different platforms, and it will be annotated with platform specific information such as the partition of the tuple space it has able to answer for. Annoucements to peer will be based on this schema also, and routing will thus propagate though the domains and contribute to more and more normalized schemes before converging to a stable state. Through this normalization, since we will reconstruct smaller tables from views, we will enhance the value of data and enlarge the expressiveness of our query language, being able to answer queries that we not even forethought. At the same time, this will allow for an aggregation of partitions announced by various peers. Normalization has further advantages for the manipulation of the data sets at the different platform, and we refer the reader to the large amount of work that has been conducted in this domain.

Dynamics.

While not being highly dynamic, we expect changes to occur from time to time like new data becoming available, or the appearance of a new actor. We thus need to consider an announcement protocol that will run continuously between peers, that will keep them synchronized about new routes, and those becoming invalid, just like it is done today in BGP.

Those announces will propagate though the network and schema information held by the different platforms will thus be updated and possibly refined. We remark that the choice of the third normal form for this schema has the additional interesting property of accounting for the slightest changes in such cases, and to be feasible incrementally. Such changes should not invalidate any existing queries since only eventually introduce a change in the normalized schema, but not in the views that are build on top of it and used to inform clients about the range of queries.

Policies.

We have so far considered a simple scenario where peering agreements lead both platforms to announce their full set of routes. It is possible to extend those announcements by a system of import and export policies that will selectively receive and transmit routes towards a peer. Such policy will be local to each domain and allow it to enforce some control over the traffic it expects to receive from a given peer. **References to RIBin and RIBout on the figure.**

5.3 Routing

Distributed query plane.

We now consider the answering of a query in the federation context. When the first router receives it, it inspects its routing table, in our case the normalized schema, to discover which combination of platform is able to answer the query. It then prepares a set of subqueries to those platforms, and builds and optimizes a query plane on top of them to aggregate and process incoming information. All this is done using platform metadata received from the announcements. All the contacted platforms will go through the same process until the final domains responsible for answering it has been reached.

The overall process is a distributed realization of a query plane, involving various independent and heterogeneous actors, as if the federation was a single big database. Since all routers maintain their local part of this plane internally for forwarding purposes, we end up with a query triggering the creation of a circuit that will handle the data flow coming in the reverse direction (at least in the pull-based scheme).

Pave the way to flow based traffic control. See extensions to this work section.

Dataflow – Operators – Streaming databases analogy. Dynamic and reoptimization as suggested in Section 4.3.

We need a schema with the overview

Can we make an analogy with a multicast group ?

5.4 Scalability

Only the useful part of the schema, only by the hosts that have to route and transport it. No initial knowledge of the schema, which can evolve dynamically according to the need. Routing only occurs at the edge of the network, done by the different domains. We envisage an eventual support from the network in Section 9.

dynamic aspects, reoptimization

6. APPLICATIONS

In this section, we illustrate some possible utilization of our framework through a set of representative use cases that we can find into other measurement federation proposals. They are far from being exhaustive, and we in fact expect that this architecture allows for the emergence of unforeseen new actors bringing value into the system.

Distributed applications.

partitioning: space / time share responsibility for storing different classes of measurements eventually different domains : one way of splitting among many others

collaborative monitoring

Reactive measurements.

alarm on demand DSL : trigger -> callback

communication layer between services

storage service.

Get: traceroute, callback to storage on data, create !

or find a storage place

aggregation of measurement data: measure various complementary facets of the Internet, complementary, need for reactive meausrmeent to trigger measurements form a signal

contextualization of measurement data: in testbeds, measurements about testbed resources or its substrate (eg. planetlab nodes and public internet) or user experiment: requires some deductions

diagnostic of networks (perfSONAR): end to end path, require union of various results.

real time / streaming troubleshooting, monitoring : alarms, events. heterogeneous, collaborative and automatic allow rules and trigger to describe relationships

architecture allowing to distribute horizontally and vertically (partition vs functionalities):

7. IMPLEMENTATION AND USE CASES

Could be implemented as a protocol on top of IP, or as a new network paradigm. Though for our current purpose, we implemented it as an overlay on top of the current network.

7.1 The MANIFOLD component

Implemented as an overlay to the current internet.

7.1.1 API

7.1.2 GUI

7.1.3 Authentication

7.2 OneLab services: TopHat & MySlice

8. RELATED WORK

Ontologies

Example of measurement ontologies.

perfsonar measurement ontologies (openlab, novi, etc.) IPPM

Architecture

perfsonar SOA "fixed" set of actors, even though some are generic standardize both content and implementation format complex protocols for various tasks no routing, set of lookup services user acccess everything: based on identity federation only consider partitioning of data (UNION): no joining all the complexity exposed to user. different policy architecture

SFA OMF FRCP

no formal base support for network ???? nothing to standardize about dynamic aspects of the data and actors. Nothing has to be known by the network !!!!

Semantic = end hosts,

Allow many perspectives, some are described here

9. EXTENSIONS TO THIS WORK

Formalization of the problem

Flow/Congestion control

packets protocols just like in the standard internet Could even be implemented on top of IP

Support from the network

routing at the edge, processing could be done optimally in the network benefit from many users, etc duplication query plane optimization etc.

Temporal aspects

A temporal parameter is added to the query to allow for the repatriate of existing results at a given epoch, but also to trigger on-demand measurements or even schedule new ones in the future.

Temporal consistency cache on-demand measurements Future measurement = schedule

Transparency and provenance

Security

Authentication, authorization

Economical models

What will the ecosystem look like ? On more generic objects ? how to share revenue cf panos (diversity) cf internet, AS relationships

impact on the shape (hierarchical, flat, etc.) we do not know what are incentives for people, what can have value for them.

10. REFERENCES