Routing queries in a federated environment

1. INTRODUCTION

The issue of accessing and sharing data or more generally some functionalities within a federation

Federated systems: BGP analogy one has customers, one has resources

2. REQUIREMENTS & CHALLENGES

Allow user and user tools to communicate seamlessly with various data repositories and measurement/monitoring services Allow for non-interactive interaction between

We won't distinguish measurement and monitoring for our purposes here, and will consider indistinctely information about the testbed itself, its resources, its substrate (eg. the Public internet) or a user experiment (often refered as a slice).

Different kind of measurements, from remote to local (testbed, third party platform, user associated database or local file).

Many data formats, semantic, transport, authentication, etc. and we cannot expect that we can reach a consensus since we are dealing with many autonomous entities way beyond the testbed borders (commercial services, webservices, etc.)

temporal (past, on demand, future = scheduling) callback (async channel . . . explained later)

beyond testbed reach.. need to account for multiple schemes, we cannot expect uniformization sometimes soon

2.1 Ecosystem

3. TOPHAT & MYSLICE

UPMC is running the TopHat and MySlice services, targeting two community of users, respectively the measurement and the experimenter ones. Both rely on the MANIFOLD interconnection framework that will be described in Section ??.

3.1 TopHat

Overview

The TopHat measurement service has the dual role of collecting and aggregating measurements related to the PlanetLab testbed overlay to serve both the measuremnt community, but also supports experimenters from the testbed with measurements. It offers a dedicated service that provides network topology information to PlanetLab users and applications running on PlanetLab.

TopHat supplies measurements to its users thanks to its own TDMI platform, which are supplemented by drawing upon several independent specialized measurement and monitoring infrastructures, some of which have a proven track record of excellence in providing specialized measurements to the research community.

TopHat's niche lies in its ability to support of the entire lifecycle of an experiment: assisting users during experiment setup in choosing the nodes on which the experiment will be deployed; at run time, it providing live information to support adaptive applications and experiment control; and supporting retrospective analysis through access to archived data.

The tool architecture (see Section ?? allows it to support in the same way the ongoing testbed federation efforts, and allow measurement and monitoring integration across the different domains. It provides both an archive of regular measurements of the topology (the testbed substrate) and of the testbed infrastructure itself.

TDMI

The TopHat Dedicated Measurement Infrastructure (TDMI) is TopHat's own measurement infrastructure. It consists in modular probing agents that are deployed in a slice of the various PlanetLab nodes and probe the underlying network in a distributed efficient manner. In addition, they probe outwards to a number of target IP addresses that are not within PlanetLab. The aim of TDMI is to obtain the basic information required by TopHat. It implements such algorithms as Paris Traceroute to remove the artifacts arising from the presence of load balancers in the Internet. TDMI aims at providing the necessary information to TopHat users about the evolution of the overlay, and focuses on catching the dynamic aspects of the topology.

More details about the available data are presented in Section ??.

Interconnected platforms

TopHat currently provides access to a set of interconnected system of different types:

measurement systems : TMDI, Gulliver, Dimes, Etomic, SONoMA

system-level monitoring infrastructure and testbed resources : CoMon, CoTop, MyPLC, Monitor, SFA

misc. information sources for topological and geographical data : Team Cymru IP to ASN mapping; MaxMind Geolite City; GeorgiaTech AS taxonomy

As TopHat relies on the modular and extensible MANIFOLD framework, it is possible to develop new platforms adapters to extend its reach.

3.2 MySlice

MySlice [MYSLICE1, MYSLICE2, MYSLICE3] consists in a user centric tool to support user's interaction with testbeds. It is tailored to support the full experiment lifecycle, from setup through completion, and it also includes the support for the management of a local authority (register and manage user accounts). Like TopHat, it is based on an open and extensible framework, as such it represents a candidate to provide a portal to the testbed federation.

The challenge MySlice solves is that the different platforms of interest for an experimenter typically realize one given functionality (SFA deals with the control plane, OMF with the experimental plane, TopHat with measurements, etc.) which is decoupled from users' needs at the various stages of their experiment. More specifically, as measurements as concerned, MySlice's objective is to put measurements into a context (measurements about a testbeds, a resource, a slice, etc.), and this is made possible thanks to the unifying framework.

MySlice currently provides the experimenter with measurements related to resources while he is selecting resources to be included into the slice, and the possibility to request new on demand measurements while the experiment is running. Finally, the possibility to collect and visualize measurements related to a slice is under development. As MySlice in itself is not the purpose of this document, the reader will refer to the referenced publications for further information about the tool.

3.3 Commonalities

• leverage a large ecosystem of available complementary and overlapping services and tools (far beyond testbed borders) • from our experience the UI is essential to users: need provide a transparent and consistent access • Exploit commonalities in platforms and processes

balance uniformity and heterogeneity

see paper FOSTER

rely on same common abstraction; MANIFOLD !

3.4 A convenience service

Convenience service, deployed for a community not mandatory, only a service that can be shortcircuited rely both on the MANIFOLD implementation we describe in the following, tailored and customized via the gateways (data and functionality) and plugins (web interface) same functionality works on user side also partially centralized service might bring additional benefits (value, latest version, etc), negociate access to sources

sufficient mass to negociate with other communities : chain of value (representative) cf Timur's talk

ihere could be multiple interconnected and interoperable deployments

3.5 An ecosystem

This step is somehow related to the efforts in federating testbeds, and should allow for an ecosystem of measurement systems to emerge, and the various actors to seamlessly exchange information. This corresponds to the vision of a distributed federation of systems (as opposed to a single central entity), where the data can be accessed through different entry points (with their own strength and specificity) and still allowing access to the full range of available information.

INDEPENDENT DOMAINS

NOTION OF VALUE data users caching processing special treatment aggregation

HOW TO SHARE REVENUE [panos]

AGREEMENTS

POLICIES

EXAMPLES OF SUCH ECOSYSTEMS

ANALOGIES WITH BGP

Status : Two communities

3.6 Collaborative and automatic monitoring

STREAMING

SEMANTIC

COMBINATION

Unlike classic approaches that rely on a set of gateways brinding the different systems, this framework helps leveraging the possibility to combine the different sources of data to enhance their value. That means we can answer user queries by a combination of two or more platforms, that were not previously available through individual platforms.

rule -> trigger

3.7 Measurement aggregation

+ convenience deployment for the measurement community : TopHat + convenient deployment for experimenters : MySlice, tight integration with the SFA federation both rely on the MANIFOLD component, which can be tailored and customized to offer the service, various interfaces : core, api and web based GUI.

3.8 Querying TopHat and associated platforms

TopHat API and web interface.

Both TopHat and TDMI data are available through an XMLRPC API hosted on https://api.top-hat.info/API, which is the priviledged way to query data. A web interface built on top of this API might provide an alternative way to browse through the measurements and visualize them.

Direct queries to a database.

Since TopHat issue calls to external platforms, it is not possible to get a direct access to a database. TopHat only uses one for caching purposes to increase the efficiency of some queries. As for TDMI, such a database exists but the high volume of measurements (full mesh Paris Traceroute measurements between all pair of PlanetLab nodes every 5 minutes) information is compressed and spread across various tables and partitions which makes the schema not directly tractable.

3.9 Exposing measurmeents to the user tools

A few words about MySlice here

Reference to the NEPI section

4. MANIFOLD: ARCHITECTURE AND COMMUNICATION PROTOCOL

We propose to define a common abstraction that will be shared by all entities in the federation, avoid avoiding the common pitfall of writing N^2 translators between them. An additional advantage of such as architecture will be to provide an appropriate layer for integrating the data originating from the different urovider in a single consistent scheme, and to extend the aggregate value beyond the value of each individual platform taken separately.

4.1 Objects and collections

In MANIFOLD, we adopted an object oriented model, which has been proved general enough to accommodate the need of the different interconnected platforms so far. A user will typically handle a **collection**, a set of objects. An **object** consists of a *class* (a type) (like slice, resource, traceroute), etc.), one or more *fields*, among which a *key* that will uniquely identify the object instance, and zero or more *methods*. A property will eventually consists in a sub-collection (a slice hold a set of associated resources, a traceroute holds a set of hops, etc.).

A **field** will consist of a name, a type/class, a flag to determine whether it is an array, a read-only or read-write flag, and a description.

4.2 Semantic

We assume that there exists an underlying semantic (or a set of semantics) that allows to precisely identify the different objects, as well as their fields and methods. It is out of the scope of MANIFOLD to deal with such aspects.

4.3 Queries and results

The interaction with the platform will be performed through a set of **queries** and **results**, which can be decoupled from platform specific aspects thanks to the use of the semantic. The basic interaction will be made possible through the 5 basic operations: `Create`, `Get`, `Update`, `Delete`[1] and `Execute`. The latter one will allow us to run methods (for starting a slice for instance). Those basic operations are both relevant to persistant storage or to user interfaces, and we remark that they form the basic operations of SQL or HTTP REST interfaces.

Results associated to a query will consist in most cases of a collection.

4.4 Building collection with relational operators

We voluntarily limit queries to a single class of objects.

We can make an analogy with the SQL query language (or more precisely the relational algebra) considering such target object form the FROM operator. This is a strategy commonly adopted in data warehousing or data integration solutions: a given information of interest for the user is represented, and augmented, annotated with various kind of information (also called facets, or dimensions). For example, a given class can be augmented with a temporal or a geographical dimension.

As it is unlikely that a single platform will be able to provide all relevant information, we might consider this is a limiting choice that will make it necessary to issue several consecutive queries. Though, we will see in the next section that this choice is not a limitation of the expressiveness of our query language but will instead bring us flexibility (in short we propose an intelligent mediator that will be able to perform join operations seamlessly in order to assemble information originating from several table, eventually cross-platform).

We also borrow from this database formalism the following set of basic operations:

- **filters** (the selection operator, or WHERE) consist in a clause of predicates. A predicate is a (key, operator, value) triplet, and a clause is an logical expression made from AND and OR operations separating clauses. Filters will allow us to build a collection;
- **fields** (the projection operator, or SELECT) enumerates a set of fields that will allow us to limit the scope of the objects being manipulated;
- **params** finally denote a set of (key,value) pairs that can be used in our queries (for updating fields when this is possible).

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.

To summarize, we denote queries as the following tuple:

```
(operation, class, filters, fields, params, ts, callback)
```

Not all elements will be needed for each operation:

operation	method	filters	params	fields	ts	callback
CREATE	✔			✔		(!)
GET	✔	✔		✔	✔	(!)
UPDATE	✔	✔	✔	✔		(!)
DELETE	✔	✔				(!)
EXECUTE	✔	✔	✔	✔		(!)

This allows us to propose the following shortcuts:

```
Get(object, filters, fields, [timestamp, [callback]])
Update(object, filters, params, fields, [callback])
Create(object, params, [callback])
Delete(object, filters, [callback])
Execute(object, filters, params, [callback])
```

The *timestamp* and *callback* elements are optional and respectively allow to specify a time range for the query (timstamp, interval, keywords), as well as a return channel (for long, asynchronous, periodic or large queries, or to be notified of some events). They will be detailed in further documents.

A query might also need to transport authentication token. This will be detailed in Section 6.3.

4.5 Platform adapters: the notion of gateways

Our framework assumes that all platform expose a consistent query interface, and result results in the expected format. Unless some of them natively conform to this specification, we propose a mechanism of adaptors, named **gateways**.

A gateway will translate queries in the format that is expected by the platform, and map results into collections. Gateways will thus be in charge of handling issues related to transport and data representation. In addition, it might also adapt the semantic used by the platform when it differs from the adopted one.

We call *metadata* the minimal set of information that has to be provided by the platform or the gateway to allow for

[1]often denoted CRUD, which stands for Create / Read / Update / Delete

federation. This of course consists in the object description, which we extend with flag informing whether the platform supports fields, filters or any other operator. This will be of particular interest in the mediator when we have to determine which query can be sent to the platform, and which treatments are to be performed locally afterwards.

In some cases (a SQL database interface) it is possible to determine metadata automatically from the platform: the geteway will this propose such a functionality.

5. INTELLIGENT MEDIATOR

In this section we illustrate how the different components work together to provide access to the different sources of data. For simplicity, we consider four platforms (A to D) which all offer a single method. A method provides access to a set of measurements, each consisting in a set of fields identified by a unique colour. This representation is somehow similar to a relational database, and allows modeling for a wide range of sources of information, from web services to flat files.

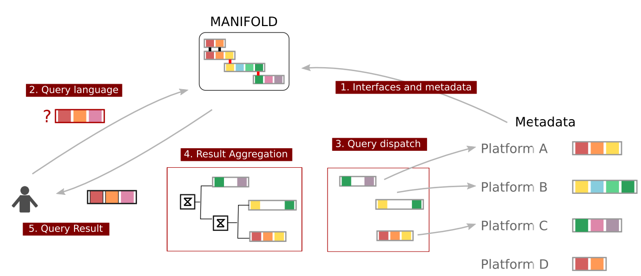

Figure 1: Illustration of a MANIFOLD query

1. Interfaces and metadata.

Each platform is described in a metadata files which is advertised to TopHat. This XML document gives general information about the platform, the way to connect to it, and the different methods and fields offered. Some fields uniquely identify a measurement and can be used for merging the data originating from several method calls, eventually from different platforms. This allows TopHat to form a global schema of available information. We note that this step requires a proper characterization of the ontologies used by the different platforms.

2. Querying TopHat.

TopHat proposes a simple query language to access the aggregated information. A query consists in a method, a temporal information, a filter pattern and a set of fields of interest, and we notice again the similarity with relational approaches. The methods, filters and fields are dependent on the interconnected platform and their availability, and it is important to notice that this range of these parameters is thus loosely defined, even though it might be possible to restrict it to a given ontology in future work.

3. Query dispatch. . . .

TopHat will issue a set of queries to the different interconnected platforms to answer the user query in an optimal way. Since there might be redundant information (we assume no conflict), the platform will automatically choose the best combination of subqueries, but will also propose ways to guide its decisions.

3. . . . and aggregation.

The result of the different subqueries will be joined together (eventually resolving time consistency issues thanks to the timing specifier of the user query), and send the result back to the user in the expected format. In addition to the fields provided by the platforms, TopHat introduces a few more ones to get information about the actual routing and aggregation performed: it is thus possible to annotate the different pieces of information with their origin, and their corresponding timestamp.

[WHY THE INTELLIGENT MEDIATOR IS GOOD FOR USERS : NO WORRY ABOUT WHO IS PROVIDING THE DATA/SERVICE, JUST QUERY FOR IT] [USED TO IMPLEMENT API and especially the GUI, GOOD FOR PLUGINS] [EXPLAIN THE DATA FLOW]

6. THE MANIFOLD IMPLEMENTATION

NOTE: there are many possible implementation of this framework

6.1 Overview

MANIFOLD provides an implementation of the different functionalities we have presented so far: standard interfaces for querying the system or individual platforms for results; a set of gateways handling various platforms; the mediator functionality that allows to dispatch and route a query to the set of appropriate platforms, extending the value of the individual platforms; support for various authentication methods; a modular and extensible interface for the gui, offering visualization of results, and allowing to balance the diversity of results we get (the real value of the federation) and a uniform-enough presentation to the user.

6.2 Interfaces

MANIFOLD proposes a wide-range of interfaces to accommodate the diversity of users' needs:

a core library (written in Python) brings most of the MANIFOLD functionality, and exposes the query interface we have presented previously. Such a library can be conveniently integrated into third party projects in order to benefit from **MANIFOLD** the advanced mediator functionality, or simply from existing adapter written for it;

an API layer (currently XMLRPC, JSON and XMPP) exposes the core functionalities remotely via the same interface to which an authentication token is added (see below);

a web frontend : for most users needs, especially new users, a graphical user frontend will be the most convenient choice. It will allow for extended interaction and visualization with the data originating from the federation. MANIFOLD currently proposes a Joomla-based PHP frontend, and a Django version written in Python is being alternatively developed. Both rely on javascript over the jQuery framework for handling dynamic content. They integrate a plugin interface that allows for the customization of the interface, and more convenient way to display platform-specific information.

6.3 Authentication

In general, the different platforms interconnected will all have different authentication schemes, and thus will require different tokens. Myslice supports multiple users and allows to store for each of them the set of necessary token to contact the different platforms.

Despite federation efforts, it is likely that support for multiple authorization schemes will remain necessary. Consider as an example the PlanetLab Europe testbed, deployed over the public internet; the user might be interested in measurements or other kind of information provided by sources run by third parties well beyond the testbed borders, and the probability that all interconnected platforms support the same authentication scheme is close to zero.

Myslice is designed to support multiple users. It can be used either locally (python library) or remotely (API or GUI). In a local setup, a user does not need to authenticate to Myslice, while this is made mandatory for remote access by the API.

The following figure succintly presents how platforms, users and accounts are handled by MANIFOLD in its internal database:

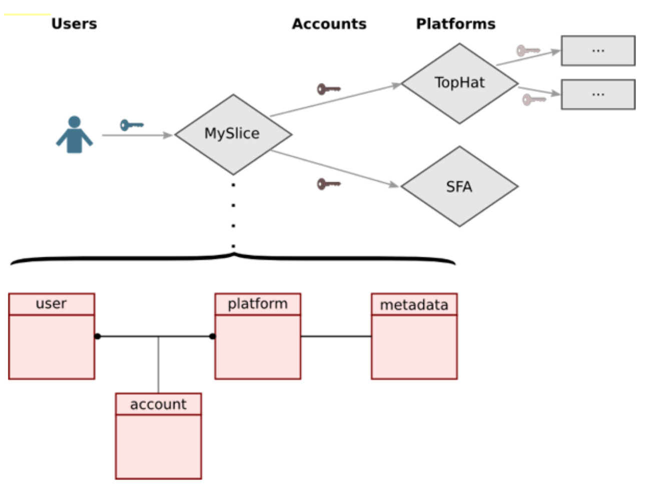

Figure 2: MySlice database

Authentication to Myslice.

This functionality is used when users are remotely accessing MANIFOLD, either through the web GUI or the API.

Currently three authentication methods are supported:

- users have a local account
- users enter their OneLab/PlanetLab Europe or PlanetLab Central authentication token
- users enter their signed SFA certificate in their browser (only supported by the web GUI at the moment)

This scheme could be extended to more advanced solutions in the future, should the need arise (eg. SSO, etc.).

Authentication and authorization to remote platforms.

For each user, a set of accounts store the necessary information to authenticate and/or get authorized on the different platforms.

Supported methods include:

- anynymous access;
- unique authentication token (handled by the system, eg a login/password);
- user authentication token (better, since it allows for accounting);
- user delegated credentials (based on SFA, priviledged mechanism allowing accounting and higher trust).

interface between user tools and measurement and heterogeneous monitoring sources tight integration into MySlice protocol and standard representation format: query/results rely on a proper semantic definition framework to propose platform adapters

6.4 GUI architecture

TODO cf Loic's section on the TRAC

Platform for development: convenient aggregation and interoperability layer between the various services and the UI - async query mechanism - transparent access to data and functions - a uthentication information - caching and query optimization

functionality/data not present/etc.

7. USAGE MONITORING IN A FEDERATED ENVIRONMENT

- reporting EU, sponsors, public
- utilization of the platform
- its value in a federation [CONEXT]
- health / debug / FLS
- long term engineering tasks
- support experiments, research about measurements itself
- events, reactive measurement, probing

[TOPHAT] [MYSLICE-FOSTER] [MYSLICE-GEC] [MYSLICE-FEW]

8. REFERENCES