Project Progress

1. Introduction

PingER has a huge amount of data and, until the conclusion of this project, the easiest way to retrieve the data is through Pingtable [ref]. Pingtable provides a friendly web interface to retrieve PingER raw data (millions of text files) and load it into a human readable HTML page. However, this is not a web standard and crossing PingER data to generate very specific information may not be possible or extremely difficult using the existing way to retrieve PingER data. This project attempts to provide a standard semantic web format to data retrievable in Pingtable.

Semantic web [ref] is a W3C recommendation [ref] and is very useful to publish data on the web. The way this project is doing it is storing PingER data in a Linked Open standard format for data interchange on the Web. More specifically, the project is converting PingER data into a graph database in Resource Description Framework (RDF) format [ref]. This graph database is very flexible; the schema can be efficiently evolved with time; and the whole data can be retrieved using a Sparql Query [ref].

The advantages of putting data in this format are that since it is a W3C standard recommendation, there is a large community familiar with it and developing research on it. Hence, it should be very comfortable to those people to retrieve PingER data using this method. If someone interested in the data has never heard about RDF, it is very simple to learn what is needed to use in the project. In addition, we are publishing data in a totally open format so anyone can use it. More importantly, since it is open, someone can come up with a never imagined interesting usage of the data.

Moreover, our data is being linked to many other existing datasets, such as DBPedia (a graph database for Wikipedia) [ref], Geonames (geographic information) ref, and Freebase [ref]. The data can also be easily linked to any other existing database published in this format. The number of initiatives publishing data in this format has been growing over time so we may also find an even better and totally different usage of this project in a near future [ref]. The graph database is obviously internally linked with its own PingER data so we can create very specific, rich, smart, and interesting queries to retrieve data that would never (or are extremely difficult to) be retrieved without the project.

Finally, the existing APIs to handle RDF provide well-known publishing formats such as JSON [ref], CSV [ref], and XML [ref]. We can conveniently get the results of a query and put it into visualization libraries to come up with very interesting visualizations of the data.

2. Ontology

a. Goal

To define the vocabulary used by PingER as well as its terms, concepts, taxonomy, and relations with each other.

b. Progress

We are basing our ontology on the one proposed by the project MOMENT[1] described by the article *Measurement Ontology for IP traffic (MOI); IP traffic measurement ontologies architecture*, by the European Telecommunications Standards Institute (ETSI) [2].

More terms and relations were added and edited on the based ontology. More specifically, metrics were properly added (Directivity, Conditional Loss Probability, Duplicate Packets, Inter-Packet Delay Variation, Out of Order Packets, TCP Throughput, Ping Unpredictability, Ping Unreachability, and Zero Packet Loss Frequency). Statistical measurements were also added (Inter-quartile Range, Median, 90%, 95%, and 75%). Information about the nodes were also added based on the Confluence specifications [3] (Site Name, Nick Name, Full Name, Contact for the host, Comments, Trace Server, App User, Ping Size, Ping Server, URL, Project Type, and GMT).

Other existing ontologies are also being linked to our ontology:

- Freebase [ref], an open large graph database.
- DBPedia [6], the ontology and the resources, will be used to provide more information about any geographic location or any other thing Wikipedia has that can be connected to the PingER. So it should be possible to make very specific queries.

c. To do

After doing very complex *mashups* [ref] using the ontology, we need to verify if the ontology needs any adjustments.

3. RDF Repository

a. Goal

To establish a good environment for the RDF repository.

b. Progress

We analyzed the existing technologies to make it possible. There are well-known Triple Storages such as Jena, Sesame, and Virtuoso [7]. According to [7], Virtuoso struggles to load large datasets (>1M triples). Hence, we first decided to try Jena SDB (with MySQL) and then Sesame Native.

i. Jena SDB results

After establishing Jena SDB 1.3.6 with a native Java API for storing and querying a relational database MySQL [ref], [ref], we were able to load a reasonable large amount of data to test.

More specifically, data of all nodes and all yearly measurement data were loaded. However, querying the dataset was very disappointing. Some very simple queries, such as listing the value of a given metric in a given year for a given pair of sites took 7 minutes to run. We only had less than 100K triples.

Since we would have much more than 100K triples and 7 minutes is way too long for such a simple query, we decided to change the RDF repository. In addition, the results of the articles [7] and [8] show that there are better alternatives. Therefore, Jena SDB may work very well for smaller datasets, which is not PingER's case.

ii. Sesame Native results

Using the results of [7] and [8], we decided to try Open RDF Sesame Native 2.7.2. This solution requires a Java Webserver as an interface with the loading program and the RDF repository itself. We are using the latest Tomcat version, Tomcat 7 [ref]. The physical layer implementation of the repository provides customization of indexes, which it is said to significantly enhance the performance of the queries execution [ref].

We are using the indexes *spoc,sopc,psoc,posc,opsc,ospc.* The order of the indexes for [s]ubject, [o]bject, and [p]redicate defines the pattern of a search. For example, the index *spoc* optimizes a search in which the subject is the first field of the statement (a statement is composed by subject, predicate, and object). More specifically, a query that would take advantage of this index would be: list all instances of metrics (the subject to be searched for) that are of the "type" "TCP Throughput". "Type" (the predicate) and "TCP Throughput" (the object) are given and we are looking for the metric (the subject). The [c] ontext is a fourth field in "triple" statements and is used as named graphs in Sesame. Our solution does not consider contexts, though.

However, multiple indexes increases the used disk space (to store the indexes) and it also takes longer to load the data to maintain the indexes.

Note: [7] points out that the largest dataset known to be loaded into Sesame Native has only 50M triples. If we extrapolate this number, there is no guarantee that the performance will remain satisfactory. Actually, there is already a warning in this scenario. We already have 12,5M triples.

Finally, the exactly same query that took 7 minutes to run, in solution (i) – Jena SDB --, took less than 5 seconds using solution (ii) – Open RDF Sesame Native. The circumstances tested were the same in both environments, i.e., same computer, same dataset, and same query.

Therefore, we decided to migrate the project to use Open RDF Sesame Native as RDF Repository.

c. To do

Run very complex queries to test the performance.

4. Accessing the RDF Repository

a. Goal

Establish an easy way to access the RDF data.

b. Progress

We are using Java Tomcat Webserver to host a Sparql Endpoint [ref]. The HTML page has a text area in which the user will be able to write Sparql 1.1 queries [ref] to access the RDF repository. By default, the results are showed in HTML tables, in a JSP page. However, there will be a combo box to choose in which format the results will be shown. Available formats will be triples in CSV, JSON, and XML-RDF.

c. To do

Use CSS and JavaScript to make it prettier.

5. Loading the RDF repository

a. Goal

Generate RDF data using external datasets and PingER data.

b. Progress

The process of generating RDF data and populating the RDF repository is divided into subsections:

i. Set up the prefixes

Following the RDF standard, all resources are uniquely identified by an URI [ref]. In order to write less and to provide a better organization of the statements, it is common to use namespaces (prefixes) instead of writing absolute URI. For example, it is common to use the namespace **rdfs** for the W3C rdf-schema (http://www.w3.org/TR/rdf-schema/).

- Frequency: If the repository is totally empty, i.e., it has just been created, the program should first insert the prefixes into the database. In other words, this should be done only once.
- Time to load: Less than 1 second.

ii. Instantiate Continents

The continents are internally instantiated in the memory of the program so there is no HTTP GET [ref].

- Frequency: Only once.
- Time to load: Less than 1 second.

iii. Instantiate Countries

The program uses a HTTP GET to access Geonames API [ref] to retrieve a JSON with data of all countries. Then, for each entry of the JSON, the program instantiates a country in RDF and loads into the repository.

- Frequency: Only once.
- Time to load: Less than 3 minutes.

iv. Generate Node Details JSON

A HTTP GET is done on PingER data to retrieve the nodes and their information (IP, Nickname, Site Name, Latitude, Longitude, etc.). A JSON is then generated (and written into a file) to be used by the program.

- Frequency: Same as the generation of the %NODE_DETAILS (about each 4 hours).
- Time to generate JSON: Less than 4 seconds.

v. Instantiate Towns and States

For each entry in the Node Details JSON, the program runs HTTP GETs on the Geonames API to try to find the nearest town (with at least 1000 habitants) and city (with at least 15000 habitants) based on the latitude and longitude of the site. The state where the town is located (if applicable) is also instantiated and linked to the town. The program finally inserts the instantiated data into the repository.

Note: The program also tries to link the existing found town with other known RDF datasets (DBPedia [ref] and Freebase [ref]).

- Frequency: *Probably* the same as (iv).
- Time to load: ~30 minutes.

vi. Instantiate Schools

For each entry in the Node Details JSON, the program runs HTTP GETs on the DBPedia Sparql Endpoint to try to find a school whose name is similar to the site's Full Name, on Node Details JSON.

If found, an instance of a school is inserted into the repository. Information about school includes endowment, number of students (undergrad and postgrad), faculty size, etc.)

Note: The program also tries to link the existing found school with Freebase.

Note2: DBPedia (just like Wikipedia) is not complete. Thus, for example, information that is available for a very famous and well-known university may not be available for another university that is not as famous.

- Frequency: *Probably* the same as (iv).
- Time to load: ~30 minutes.

vii. Instantiate Nodes

After instantiating Towns, States, Schools, Continents, and Countries, we can finally instantiate the PingER nodes.

Again, for each entry of the Node Details JSON, the program instantiates a Node linking it with its respective Town, School (if it is one), Country, State (if applicable), and Continent.

- Frequency: *Probably* the same as (iv).
- Time to load: Less than 3 minutes.

c. To do

Make better (more precise and informative) annotations of how long each step above takes to run. Maybe space utilized should also be included.

Complex tests should be performed.

Study other forms of retrieving information about schools.

Study ways to optimize this process. Maybe parallelizing?

6. Loading the RDF Repository (PingER Measurements)

a. Goal

Although this step remains in the same line of the previous one (5), it can be separated into a totally different context – the measurements context, just as a matter of better organization and understanding. The goal is to generate RDF data from PingER dataset and load it into the repository.

b. Progress

After loading the repository with the data specified in the previous step (5), the program needs to load PingER measurement data.

The first step in this process is to generate the Monitoring-Monitored [ref] JSON. The program executes a HTTP GET in http://www-wanmon.slac.stanford.edu/cgi-wrap/dbprac.pl?monalias=all to retrieve all monitoring nodes. Then, for each monitoring node, another HTTP GET is executed in http://www-wanmon.slac.stanford.edu/cgi-wrap/dbprac.pl?monalias= EDU.SLAC.STANFORD.N3&find=1, where the value of *monalias* is a given monitoring node, to retrieve the monitored nodes by that monitoring node. The Monitoring-Monitored JSON is then generated and written into a file.

- Frequency of generating this JSON: *Probably* the same as (5.iv)
- Time to generate: Less than 2 minutes.

Having the JSON, the instantiating process happens according to this approach: For each monitoring node (entry of the JSON), for each metric, for each packet size, for each time parameter, the program executes a HTTP GET in the Pingtable [ref] Tab Separated Values (TSV) file specified by crossing all these parameters. A TSV URL example is of the form

http://www-wanmon.slac.stanford.edu/cgi-wrap/pingtable.pl?format=tsv&file=average_rtt&by=by-node&size=100&tick=allyearly&from=EDU.SLAC.STANFORD.N3&to=WORLD&ex=none&only=all&dataset=hep&percentage=any

Where the parameters:

- from – is the monitoring node that pings other monitored nodes.
- tick – represents the time aggregation. PingER has data from 1998 to 2013. At this moment, the project is considering only the following tick parameters:
    - allyearly
    - allmonthly
    - last365days
- size – is the packet size. At this moment, the project is considering only packets sizes of 100 bytes.
- file – is the network measurement metric. At this moment, the project is considering only the following metrics:
    - Mean Opinion Scores
    - Directivity
    - Average Round Trip Time
    - Conditional Loss Probability
    - Duplicate Packets
    - Inter Packet Delay Variation
    - Minimum Round Trip Delay
    - Packet Loss
    - TCP Throughput
    - Unreachability
    - Zero Packet Loss Frequency

Note: This process is totally independent of the previous step (5). Hence it can be independently parallelized. However, if this step is executed before the previous, the measurement information regarding the nodes will point to broken links, which is not a big problem and does not prevent loading measurement data. The broken links will be automatically repaired when the nodes are successfully instantiated (section 5.vii).

c. Performance Evaluation

Last 365 days: For each monitoring node, for each metric, it is taking around 1 hour to load the data into the repository. Hence, for 80 monitoring nodes, for the 11 metrics, it is going to take approximately 880 hours (36 days). **Thus, impracticable amount of time.**

d. To do

It is taking a huge amount of time to load the entire data. We must optimize this process. The project is being adjusted to be easier parallelizable in independent processes and each process is being investigated to try to gain some minutes.

Measure time and space taken to load the repository.

Complex tests.

7. Rich Visualization of the Data

a. Goal

Provide smart and useful visualization of PingER data in RDF format.

b. Progress

We studied to possibility of using 3 APIs:

i. Google Maps JavaScript API v3 [ref]
ii. Google Geo Charts [ref]
iii. Google Public Data Explorer [ref]

All of them seem to be very useful and can provide rich visualizations. (i) and (iii) seem to be the most powerful of them.

c. To do

We need to think about good and useful mashups to use within this entire project and showing them on these visualization APIs. One type of mashup that is being investigated is retrieving data from DBPedia to cross PingER data with information related to universities (such as endowment, number of students, if the university is public or private, etc).

8. Documentation

a. Goal

Document the entire project.

b. Progress

This Project Progress is being built. It is being kept in both MS WORD format and HTML (to be used in Confluence).

The Confluence page *To do-Doing-Done* is kept updated more frequently.

c. To do

The Java Project should be documented. Javadocs are supposed to be generated for each class and method.

An interactive JavaScript document is to be generated to graphically represent the ontology in order to support users to use the RDF data.

Installation guide should be written. This should include how to configure the environment and everything needed to compile and run the project. Both the RDF Repository (with Tomcat settings) and the Sparql Endpoint projects should have an installation guide.

9. References

The reference section needs to be fully reviewed. The proper pointers need to be referred in the text above, especially when the [ref] word is cited in the text.

[1] Project MOMENT Ontologies. Retrieved from https://svn.fp7-moment.eu/svn/moment/public/Ontology/ on June 5, 2013.

[2] ETSI. Measurement Ontology for IP traffic (MOI); IP traffic measurement ontologies architecture. Retrieved from http://www.etsi.org/deliver/etsi_gs/MOI/001_099/003/01.01.01_60/gs_moi003v010101p.pdf on June 5, 2013.

[3] PingER Node Details. Retrieved from https://confluence.slac.stanford.edu/display/IEPM/PingER+NODEDETAILS on June 5, 2013.

[4] Food And Agriculture Organization Of The United Nations Ontology. Retrieved from http://www.fao.org/countryprofiles/geoinfo/geopolitical/ on June 5, 2013.

[5] Friend of a Friend Ontology. Retrieved from http://www.foaf-project.org/ on June 5, 2013.

[6] DBPedia. Retrieved from http://dbpedia.org/About on June 5, 2013.

[7] Comparison of Triple Stores. Retrieved from http://www.bioontology.org/wiki/images/6/6a/Triple_Stores.pdf on June 5, 2013.

http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html

http://jena.apache.org/documentation/sdb/

http://openrdf.callimachus.net/sesame/2.7/docs/users.docbook?view#chapter-console

http://www.w3.org/TR/sparql11-query/

https://developers.google.com/chart/interactive/docs/gallery/geochart

http://www.google.com/publicdata/directory

https://developers.google.com/maps/documentation/javascript/

https://tools.ietf.org/html/rfc3986

http://www.w3schools.com/tags/ref_httpmethods.asp

http://www.geonames.org/export/web-services.html