Resource Oriented Architecture: A New Approach with REST and Graph Database

Jagadeesh Hanumantegowda BE,MS
Chief Technology Officer, FireFlink Pvt Ltd
IndiQube South Mile, Basavanagudi, Bangalore, Karnatka, India
h.jagadeesh@gmail.com

Abstract — The amount of data that's being created and stored on a global level is growing exponentially. Now with Machine-to-Machine communication and Internet of things, the amount of data generated and shared across different devices and applications over the network is huge. The essence of this paper is to show that Graph Database can be applied in prominent for capturing and processing data generated in Machine-to-Machine communication or Internet of things applications, thereby allowing the system architecture to adhere to Resource Oriented Architecture and hence supporting the unstructured and semi structured data to be exposed as resource to other applications in the distributed environment. Most of these machines generated data being either unstructured or semi structured, It is highly impossible to store and process this data in traditional RDBMS and also it requires efficient real time data collection and faster execution techniques to meet the most of the real world Machine to Machine and Internet of Things use cases. Once the data is collected and stored we need modern techniques to query and filter in order to cater to different application requirement hence making it complicated to achieve Resource Oriented Architecture in a distributed environment when involving unstructured and semi structured data.

Index Terms—ROA, Big Data, REST, Graph Database

INTRODUCTION

With semi structured and unstructured data becoming more and more prominent in all applications, middleware software in n-tier architecture has to take huge load of data manipulation before being exposed over REST interface. This data manipulation is straightforward when the data is stored in traditional RDBMS. This is because we can query and filter the data at the data base level leaving very little to middleware. Conventional query techniques are not readily available due to loosely defined schema of semi structured data. With XML and JSON becoming new standard for data exchange there has been an increased volume of data which needs to be queried. This has resulted in number of new techniques for storing and processing queries that result in XML or JSON format.

The complex nature of the semi structured data and the corresponding query processing makes query evaluation very hard. In order to store such data, we can decompose the data into different nodes and the relationship between the nodes in a a graph pattern. In traditional databases relationships are built at query time, hence resulting in expensive query processing. This should be replaced by a database which stores relationships also as a core data component. This will improve the processing of those already persistent relationships in a more efficient way. Since the relationships are also persisted, query time is a constant-time operation.

ROA: A LIGHTWEIGHT APPROACH

From a programming point of view, REST is a lightweight alternative to Web Services and RPC. Retaining this simple architectural design while exposing a semi structured data is a new challenge. Below key components of a REST architecture should be intact.

- **Resources**, which are identified by standard URLs. State and Functionality both are represented using resources. The URLs imply that the resources are universally addressable by other parts of the application or other applications. Resources are the key element of a true ROA design, and the resource should contain all the required information about the state of the object it is representing.
- **Hypermedia as the Engine of Application State**, meaning that a single resource should not be very large and contain too fine-grained details. Whenever relevant, a resource should contain links to additional information -- just as in web pages. Below is a small example HTTP response which adhere to Hypermedia as the Engine of Application State

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...
 <?xml version="1.0"?>
<account>
<account_number>12345</account_number>
 <balance currency="usd">100.00</balance>
  <link rel="deposit"
href="http://somebank.org/account/12345/deposit" />
<link rel="withdraw"
href="http://somebank.org/account/12345/withdraw" />
<link rel="transfer"
href="http://somebank.org/account/12345/transfer" />
<link rel="close"
href="http://somebank.org/account/12345/close" />
</account>
```

The system will be a client-server model, but of course one component's server can be another component's client the interaction is stateless (although the servers and resources can of course be stateful). Each new request should carry all the information required to complete a request, and must not rely on previous request with the same client.

Resources should be cacheable whenever possible The protocol must allow the server to explicitly specify which resources may be cached, and for how long. Since HTTP is universally used as the REST protocol, the HTTP cache-control headers can be used for this purpose. Clients must adhere to the server's cache specification for each resource. Additional servers can be used as part of the architecture, to improve performance and scalability.

GRAPH DATABASE: STOREING SEMI STRUCTURED DATA

Graph database is designed to model and navigate data, with extremely high performance with connected nodes and relationships. Graph database contains connected entities and relationships. Entities can hold any number of attributes (key-value-pairs). Nodes can be tagged with labels representing their different roles. Each label is like a new table in traditional database. A node as such represents row and the attributes in the node are analogous to columns.

Relationships provide relevant connections between two entities. Each relationship always has a direction, a type, a start node, and an end node. Like nodes, relationships can have any attributes. Two nodes can share any number or type of relationships and they are directed. All relationships can always be navigated regardless of direction. There should not be any broken links in graph database. This is because a relationship always has a start and end node. Deleting a node without also deleting its associated relationships is prohibited. And an existing relationship will never point to a non-existing endpoint.

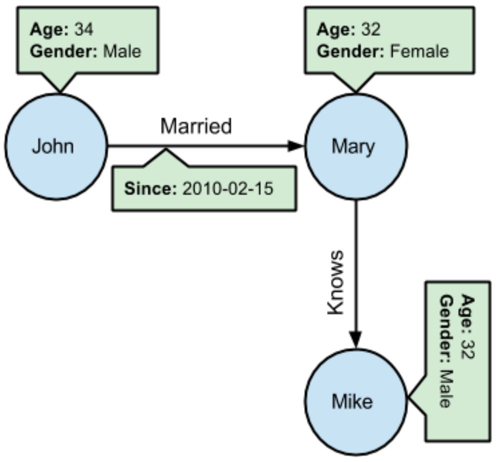

Figure 1 Graph Database Example

REST API TO GET DATA FROM GRAPH DATBASE

Easiest way to interact with Graph Database is by using REST endpoints on the data that is being exposed by Graph Database. This is not possible on the fly. There has to be a separate wrapper on Graph Database which can act as persistence service in order to expose data on REST end points. Graph Database transactional endpoint can execute db query commands within the scope of a transaction. The transaction can be kept open across multiple REST requests, until the client chooses to commit or roll back. Each REST request can include multiple statements, and can include statements along with a request to begin or commit a transaction. REST API also supports authorization and authentication. When authorization and authentication is enabled, requests to the REST API must be authorized using the username and password of a valid user. All responses from the REST API can be transmitted as JSON, resulting in better performance and lower memory overhead on the server side. Relationships are also direst resources in Graph Database REST API. Each relationship can be accessed either as a stand-alone or through the nodes it is attached to. The node intensity is the number of relationships associated with a node. Graph database can store the intensity for each node, making this a useful mechanism to quickly get the number of relationships a node has. It is also possible to filter intensity by direction and/or relationship type.

Below request and response shows how REST API can be used to get all relationships.

```
            REQUEST:
GET http://localhost:port/db/data/node/{nodeid}/relations
hips/all
Accept: application/json; charset=UTF-8

            RESPONSE:
            [ {
 "start" : "http://localhost:port/db/data/node/{id1}",
             "data" : { },
  "self" : "http://localhost:port/db/data/relationship/{id1}",
 "property" :
"http://localhost:port/db/data/relationship/{id1}/properties
/{key}",
 "properties" :
"http://localhost:port/db/data/relationship/{id1}/properties
",
            "type" : "HATES",
            "extensions" : { },
  "end" : "http://localhost:port/db/data/node/{id1}",
              "metadata" : {
               "id" : {id1},
               "type" : "HATES"
              },
            {
 "start" : "http://localhost:port/db/data/node/{id2}",
             "data" : { },
  "self" : "http://localhost:port/db/data/relationship/{id2}",
 "property" :
"http://localhost:port/db/data/relationship/{id2}/properties
/{key}",
 "properties" :
"http://localhost:port/db/data/relationship/{id2}/properties
",
            "type" : "HATES",
            "extensions" : { },
  "end" : "http://localhost:port/db/data/node/{id2}",
              "metadata" : {
               "id" : {id2},
               "type" : "HATES"
              }
              ]
```

Schema is optional in graph database. But is possible to introduce schema in order to gain performance or modeling benefits. Graph database also supports indexing. Simple example request and response to create an index using REST API is as shown below.

```
        REQUEST:
POST http://localhost:port/db/data/schema/index/label_1
Accept: application/json; charset=UTF-8
Content-Type: application/json
        {
"property_keys" : [ "property_1" ]
        }

        RESPONSE:
200: OK
Content-Type: application/json; charset=UTF-8

{
 "label" : "label_1",
 "property_keys" : [ "property_1" ]
}
```

In Graph Database traversals are performed from a start node. The traversal is controlled by the URI and the body sent with the request. In order to decide how the graph should be traversed, some parameter can be sent in the request body. To progress to the subsequent page of traversal results, the client can issue a HTTP GET request on the paged traversal URI which causes the traversal to fill the next page. If the queried data is too large to return over HTTP protocol, index and offset can be still used as parameter and pagination can be achieved.

Conclusion

REST along with Graph Database can significantly change the design approach of the Resource Oriented Architecture. This in particular will be very effective when the underlying resources are derived from semi structured or unstructured data. Having a persistence service wrapper on top of Graph Database will allow applications to directly expose semi and unstructured data in XML or JSON format to other applications in distributed environment through REST APIs. Storing semi structured or unstructured data in Graph Database also has an advantage of having relationships defined well in advance while storing the data hence REST API can also expose the relationships.

This approach of having Graph Database in backend and REST APIs in the north bound with a very generic persistence service wrapper can make the middleware layer in an n-tier architecture a pass through layer and bring a whole new outlook to Resource Oriented Architecture even when the underlying resources are stored in the Graph Database from semi structured or unstructured data. Hence REST along with Graph Database will bring a new approach to the way Resource Oriented Architecture is achived.

References

[1] Treanor, Jill. Ultra-fast trading blamed for 'flash crash'. The Guardian. [Online] 8 July 2011. http://www.guardian.co.uk/business/2011/jul/08/ultra-fast-trading-blamed-for-flash-crash.

[2] Acuna, Antonio. Linked data for executives: building the business case. London : British Computer Society, 24 November 2011.

[3] Berners-Lee, Tim. Talks: Tim Berners-Lee on the next web. TED. [Online] February 2009. http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html.

[4] Resource description framework. W3C semantic web. [Online] 2 October 2004. http://www.w3.org/RDF/.

[5] Lovinger, Rachel. RDF and OWL: A simple overview of the building blocks of the semantic web. Slideshare. [Online] December 2007. http://www.slideshare.net/rlovinger/rdf-and-owl.

[6] Berners-Lee, Tim. Giant Global Graph. Massachusetts Institute of Technology Decentralised Information Group. [Online] 21 November 2007. http://dig.csail.mit.edu/breadcrumbs/node/215.

[7] Herman, Ivan. Web Ontology Language (OWL). W3C Semantic Web. [Online] 15 October 2007. http://www.w3.org/2004/OWL/.

[8] W3Schools. RDF Tutorial. W3Schools. [Online] http://www.w3schools.com/rdf/.

[9] Menday, Roger. A perspective on DaaS. s.l. : Fujitsu Laboratories of Europe Limited, 4 October 2011.

[10] Cyganiak, Richard and Jentzsch, Anja. Linking Open Data cloud diagram. [Online] September 2011. http://lod-cloud.net/.

[11] RDF Query Language. Wikipedia. [Online] http://en.wikipedia.org/wiki/RDF_query_language. [12] About data.gov.uk. Data.gov.uk. [Online] http://data.gov.uk/about.

[13] Linked data. Data.gov.uk. [Online] http://data.gov.uk/linked-data.

[14] Ralmond, Yves, et al. Case study: use of semantic web technologies on the BBC web sites. W3C semantic web use cases and case studies. [Online] January 2010. http://www.w3.org/2001/sw/sweo/public/UseCases/BBC/.

[15] Mendes, Pablo. About DBpedia. DBpedia. [Online] 8 November 2011. http://dbpedia.org/About.

[16] Wallis, Richard. WikiData - announcing Wikipedia's next bg thing. Data Liberate. [Online] 7 February 2012. http://dataliberate.com/2012/02/wikidata-announcing-wikipedias-next-big-thing/.

[17] Zaino, Jennifer. The power is in the link. semanticweb.com. [Online] 6 January 2012. http://semanticweb.com/the-power-is-in-the-link_b25765.

[18] Serbu, Jared. Navy struggles to find the way ahead on big data. Federal News Radio. [Online] 20 February 2012. http://www.federalnewsradio.com/?nid=412&sid=2754767.

[19] Menday, Roger, et al. Linked IT - the BigGraph Concept. s.l.: Fujitsu Laboratories of Europe Limited, 2011.

[20] Lohr, Steve. The age of big data. The New York Times. [Online] 11 February 2012. http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html?_r=1&pagewanted=all.

[21] Patil, DJ. Building data science teams. O'Reilly Radar. [Online] 16 September 2011. http://radar.oreilly.com/2011/09/building-data-science-teams.html.

[22] Watters, Audrey. Scraping, cleaning, and selling big data. O'Reilly Radar. [Online] 11 May 2011. http://radar.oreilly.com/2011/05/data-scraping-infochimps.html.

[23] DeWitt, David J. Big data - what is the big deal? Professional Association for SQL Server. [Online] 14 October 2011. http://www.sqlpass.org/summit/2011/Live/LiveStreaming/LiveStreamingFriday.aspx.

[24] Rodriquez, Alex. RESTful web services: the basics. IBM DeveloperWorks. [Online] 6 November 2008. https://www.ibm.com/developerworks/webservices/library/ws-restful/.