Multiple ALTO Resources Query

draft-zhang-alto-multipart-00

J. Jensen Zhang
Dawn Chan

July 16, 2018@IETF 102
Motivation

- A lot of use cases require to query multiple ALTO information resources in a single request for some reasons:
  - Efficiency
  - Consistency
  - Relational Query

- This document lists three potential use cases:
  - Simple Batch Query
  - Properties Constrained Query
  - Path Vector Query (from draft-ietf-alto-path-vector)
Additional Requirements

• The base ALTO protocol cannot work in these cases. To make them work, there are some additional requirements raised:
  – Req 2: General query schema to filter arbitrary ALTO information resource.
  – Req 3: Relational query amongst multiple ALTO information resources.
Roadmap

• This document propose a new ALTO service called ALTO "Multipart Query" service to fit these requirements.

• The overview of the techniques adopted to address each requirement:
  – Req 1 -> HTTP "multipart/related" message
  – Req 2 -> Unified request format
  – Req 3 -> General-purpose query languages, e.g., XQuery, JSONiq, etc.
HTTP "multipart/related" Message

• The response of a Multipart Query service is an HTTP message with the "multipart/related" content-type.

• Example:

  Content-Type: multipart/related; boundary=query-boundary

  --query-boundary
  Content-Type: <content-type of resource 1>

  <response data entry of resource 1>

  ... response for resource 2 – n-1 ...

  --query-boundary
  Content-Type: <content-type of resource n>

  <response data entry of resource n>
Unified Request Format

- The request format of a Multipart Query service is independent from any ALTO information resource

- Request Format:
  ```
  object {
    ResourceQuery resources<1..*>
    [JsonString query-lang]
  } ReqMultipartQuery;
  ```

  ```
  object {
    JsonString resource-id
    [JsonValue input] // POST-mode required
  } ResourceQuery;
  ```

- The "input" field is extended and can support the input parameters of any ALTO information resource.
General-purpose query languages

• The request of a Multipart Query service can support program written in general-purpose query languages.
  – The value of "input" field can be a query program.
  – The result of the query program is the input parameters for a requested ALTO information resource.
  – The query program can use the response data of another requested ALTO information resource.

• Example:

```json
{
  "query-lang": "jsoniq",
  "resources": [  
    { "resource-id": "propmap-location" }, // GET-mode for resource_0
    { "resource-id": "my-default-costmap", // POST-mode for resource_1
      "input": `let $propmap := collection("propmap-location")
        .("property-map") // Read response of resource_0 into a variable
        return { ... Construct Input Parameters ... }` ]
}
```
Logical Workflow inside the Server

Dependency DAG

Request
- RawReq1
- RawReq2
- RawReq3

Dependency Resolver

RawReq1
- Resource1
- Query Language Parser
  - Req2
  - Resource2

RawReq2
- Query Language Parser
  - Req2
  - Resource2

RawReq3
- Resource3
- Query Language Parser
  - Req3
  - Resource3

Multipart Message
- Resource1
- Resource2
- Resource3
A Comprehensive Example

POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Length: ###
Content-Type: application/alto-multipartquery+json

{
    "query-lang": "jsoniq",
    "resources": [
        {
            "resource-id": "endpoint-path-vector",
            "input": {
                "cost-type": {
                    "cost-mode": "array",
                    "cost-metric": "ane-path"
                },
                "endpoints": {
                    "srcs": [ "ipv4:192.0.2.2" ],
                    "dsts": [ "ipv4:192.0.2.89",
                                "ipv4:203.0.113.45" ]
                }
            }
        },
        {
            "resource-id": "propmap-availbw",
            "input": `
let $propmap :=
    collection("endpoint-path-vector")
    .("endpoint-cost-map")
return {
    "entities": [
        distinct-values(flatten(
            for $src in keys($propmap)
            let $dsts := $propmap.$src
            return flatten(
                for $dst in keys($dsts)
                return $dsts.$dst
            )))
    ],
    "properties": [ "availbw" ]
}"
        }
    ]
}
A Comprehensive Example

HTTP/1.1 200 OK
Content-Length: ###
Content-Type: multipart/related; boundary=path-vector-query

--path-vector-query
Content-Type: application/alto-endpointcost+json
{
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89": [ "ane:L001", "ane:L003", "ane:L004" ],
      "ipv4:203.0.113.45": [ "ane:L001", "ane:L004", "ane:L005" ],
      "ipv6:2001:db8::10": [ "ane:L001", "ane:L005", "ane:L007" ]
    }
  }
}

--path-vector-query
Content-Type: application/alto-propmap+json
{
  "property-map": {
    "ane:L001": { "availbw": 50 },
    "ane:L003": { "availbw": 48 },
    "ane:L004": { "availbw": 55 },
    "ane:L005": { "availbw": 60 },
    "ane:L007": { "availbw": 35 }
  }
}
Remaining Issues

• Is it general enough for potential use cases?
  – Current relational query is a Pipeline/DAG
  – Can we achieve general relational query, e.g., "join" operator for relational database. Do we need this?

• Implementation complexity to support query languages.

• Security considerations to support query languages.

• Error handling:
  – How about the query program execution failed?
  – How about the query program result has pre-defined ALTO errors, e.g., E_SYNTAX, E_MISSING_FIELD, E_INVALID_FIELD_TYPE and E_INVALID_FIELD_VALUE?
Next Steps

- Implement it and show some experimental result by next meeting.
- WG item?

- See https://github.com/openalto/alto-multipart for details.
Backup Slides