Dydra
& Temporal RDF

v/s

Volume
Variety
Velocity

http://dydra.com

mepdaw 2023

SPARQL Statement Annotations
for
Temporal Metadata
in the
Dydra RDF Store

A Report on Dataset Statistics
and
Implementation Insights

1. **intent**
2. architecture
3. space
4. access time
5. annotation
6. issues ...

Big-Data implementations must, in principal, contend with (at least) the three Vs:

- volume
- velocity
- variety

Dydra has been developing and deploying RDF technology since 2010 without conscious attention to those issues, simply by addressing customer use cases with available engineering,

This workshop contribution will describe how the service approaches "archival" data management - from the implementation to a practical application and, along the way, draw insights from the ways it has been influenced by those issues over the past decade and evolved to address them.

It discusses

- architecture for dataset storage in general and versions in particular
- storage characteristics : size, access time
- access patterns : updates, queries

It describes how the service features combine to support a use case in enterprise product life cycle management

1. intent
2. **architecture**
3. space
4. access time
5. annotation
6. issues ...

Our initial intent was to serve as "github" for data.
That is, to serve as a platform upon which a developer community shared data, built analytical applications upon it, and curated their evolving data over time.

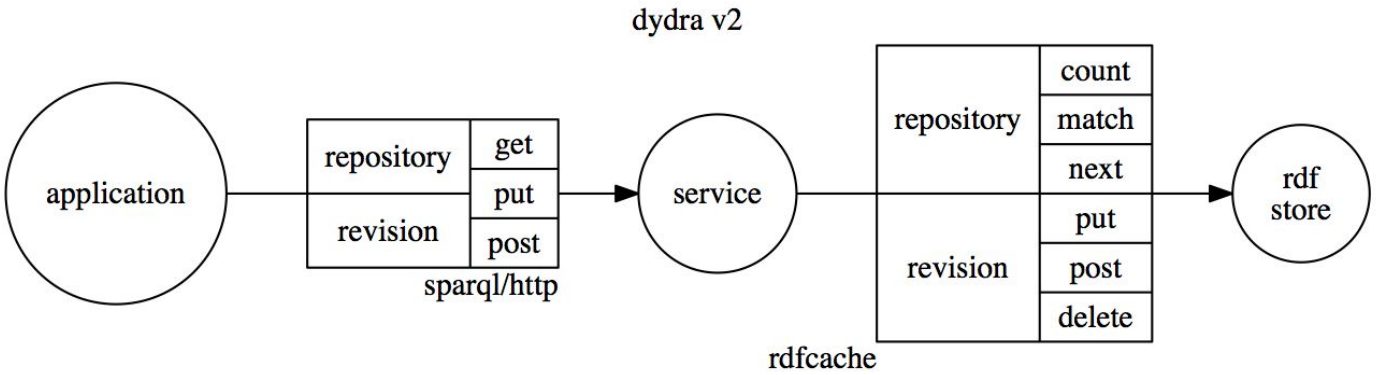
An initial naive approach based on "independent copies" offered the advantage that deduplication among repositories meant innumerable private copies of dbpedia would incur no cost.

On the other hand, we developed early-on the ability to federate among local repositories at zero-cost, and their divergent modifications suggested some sort of "change-based" approach. That version did not survive the initial experience with system recovery and upgrades.

In addition, use cases appeared which involved frequent updates to production datasets. These argued for a "timestamp-based" schematic storage model.

The uses have evolved over time to require not just local static data, but federated access to distributed, dynamic semantic data and the service has evolved to accommodate that, to provide a uniform architecture for

- streaming data
- universal tokenization
- temporally qualified analytics



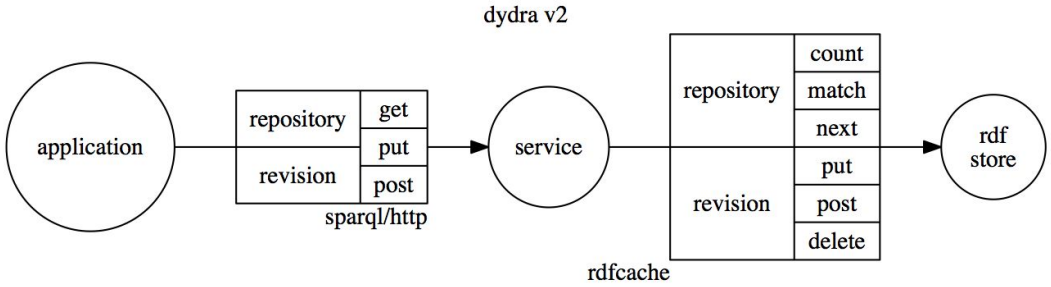
The initial version was an HTTP service to provide access to an RDF store which provided just two access patterns

- statement level access to quad data through just three operations
  - count
  - match
  - next
- repository level access
  - put
  - post
  - delete

1. intent
2. **architecture**
3. space
4. access time
5. annotation
6. issues …

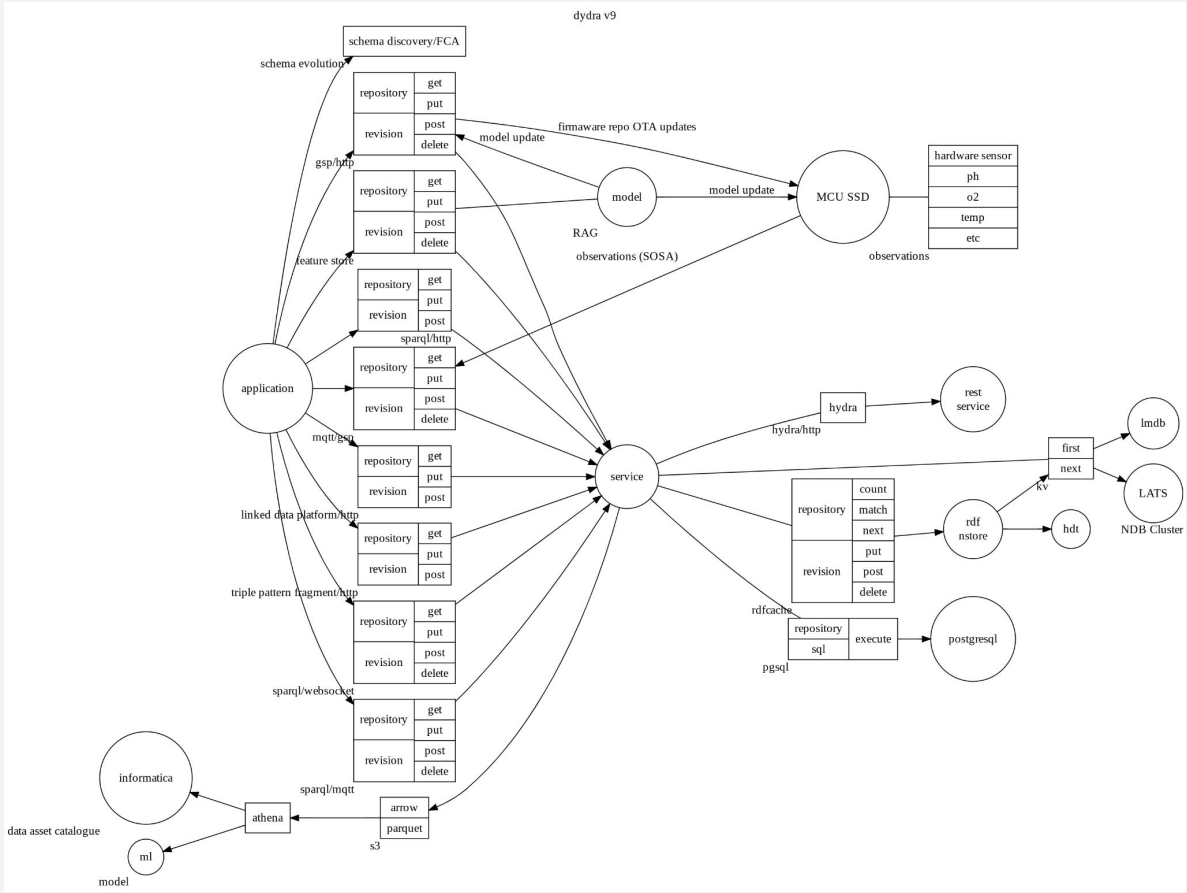
we tend to realize things a simply as possible.

That means, literally.



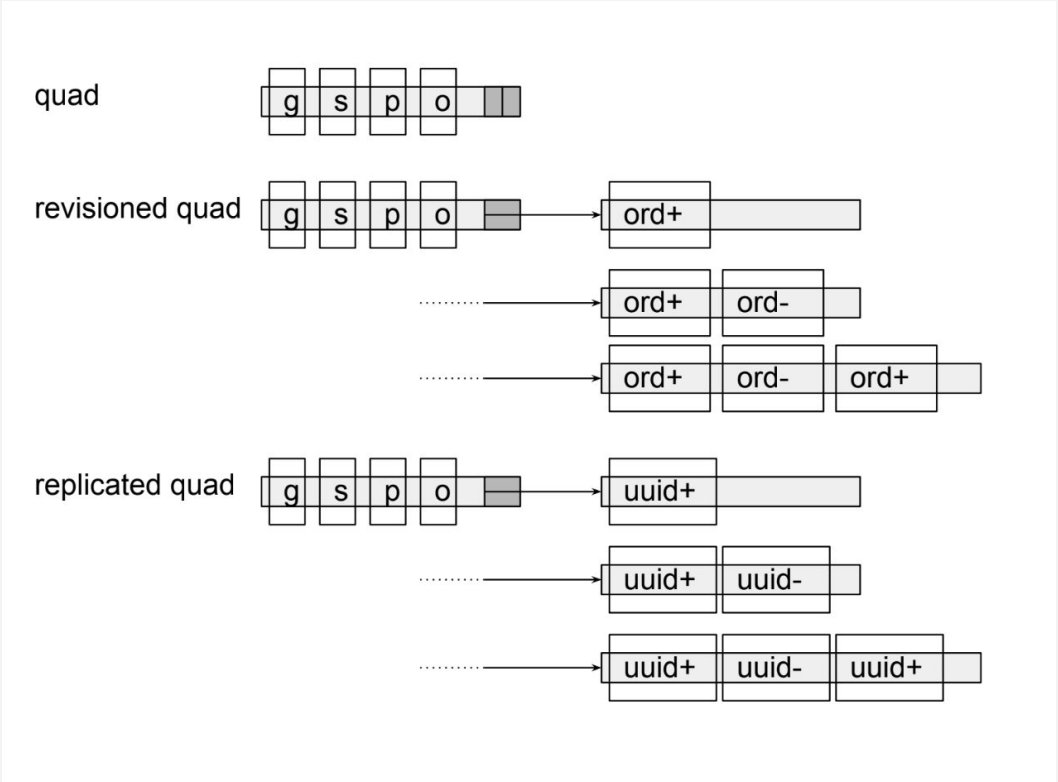

```
(defmethod rlmdb:map-repository-statements
           (operator (index rlmdb:index-database) (quad-pattern t) &key . . .)

  (labels ((get-quad (get-op)
                . . .
              (let ((return-code (liblmdb:cursor-get %cursor raw-key raw-value get-op)))
                 (alexandria:switch (return-code)
                    (0 (call-with-quad-entry raw-key raw-value))
                    (liblmdb:+notfound+ nil)
                    (t (lmdb::unknown-error return-code)))))
             (call-with-quad-entry (k v)
               . . .
               (let* ((%index-quad (%mdb-val-data k))
                 . . .
                 (map-repository-statements-callback
                    operator
                    (term-number-key-to-term-number-quad %index-quad %result-quad quad-map)))
                 (incf match-count)))))
   . . .
   )
```

1. intent
2. **architecture**
3. space
4. access time
5. annotation
6. issues ...



This evolved in response to use cases, but has retained the abstract access pattern and the W3C service interface, even as we have extended the back-end to operate on a wide variety of data sources.

1. intent
2. **architecture**
3. space
4. access time
5. annotation
6. issues ...



The match/count/scan interface applies to quads through the standard six S-P-O-G indices.
This we augmented with a linear revision ordinal index to record statement visibility in terms of identifiers for insertion and removal revisions.

The schematic structure suggests a fourth big data variable not seen elsewhere: "viability".
That is, how much implementation and maintenance effort does the capability require?

To explore that in terms of complexity, look at two things :

- changes to component logic
- incidental lines of code

Five logical operations are necessary to support this capability :

- `compute-repository-revision`
- `parse-compound-revision-designator`
- repository metadata api: `find-revision-record`, `put-revision-record`
- `map-repository-statements`
- `%test-visibility-range`

Of these, just the last two are new.
As query components are compiled and cached, their accuracy depends on the repository state, which is identified by revision.

On one hand, this issue is incidental, but its relevance will become apparent in the context of the red-line use case.

Five logical operations are necessary to support this capability . . .

- map-repository-statements
  Generate the statements which match a pattern for use in BGP processing

```
(let* ((%index-quad (%mdb-val-data k))
       (visibility-bytes (%mdb-val-size v))
       (visibility-count (if (plusp visibility-bytes) (/ visibility-bytes visibility-unit) 0)))
   (cond ((and named-only (= (cffi:mem-aref %index-quad 'term-id quad-graph-index) #xffffffff))
          ;; skip
          t)
         ((or wild-pattern-p (%quad-match-p %quad-pattern %index-quad)) ;; iff still in range
          (incf scan-count)
          (cond ((or (zerop visibility-count) ; not revisioned
                     (and (oddp visibility-count) (>= first highest)) ; test for head
                     (%test-visibility-range first last (%mdb-val-data v) visibility-count))
                 ;; if visible apply op and return the yes/no continue indication
                 (map-repository-statements-callback
                    operator
                    (if (= index-db-index 0)
                        %index-quad
                        (term-number-key-to-term-number-quad %index-quad %result-quad quad-map)))
                 (incf match-count))
                (t
                 ;; otherwise skip
                 t)))
         ;; otherwise end
         (t nil)))
```

Five logical operations are necessary to support this capability . . .

- %test-visibility-range -> %test-visibility
  Determine whether a statement is visible in a given revision.

```
(defun %test-visibility (ordinal %vector length)
  (labels ((test-ordinal (test-position)
             (let ((test-ordinal (cffi:mem-aref %vector :uint32 test-position)))
               (cond ((< test-ordinal ordinal) -1)
                     ((> test-ordinal ordinal) 1)
                     (t 0))))
           (binary-search (start end)
             (let* ((test-position (ash (+ start end) -1))
                    (result (test-ordinal test-position)))
               (ecase result
                 (0 test-position)
                 (-1 (if (= test-position start) start  (binary-search test-position end)))
                 (+1 (if (= test-position start) nil (binary-search start test-position)))))))
    (case length
      (0 t)
      (1 (when (>= ordinal (cffi:mem-aref %vector :uint32 0))
           (values t 0)))
      ;; for a very small repository, this special case was worth 0.17 elapsed time
      (2 (when (and (>= ordinal (cffi:mem-aref %vector :uint32 0))
                    (< ordinal (cffi:mem-aref %vector :uint32 1)))
           (values t 0)))
      (t (let ((position (binary-search 0 length)))
           (when position
             (cond ((evenp position)
                    (values t position))
                   ((and (oddp position) (> length (1+ position)) (= ordinal (cffi:mem-aref %vector :uint32 (1+ position))))
                    (values t (1+ position)))
                   (t
                    (values nil position)))))))))
```

Five logical operations are necessary to support this capability . . .

- `compute-repository-revision`
  Derive the revision instance given a designator.
  The core is just direct keyed retrieval (`find-revision-record`), but to interpret the compound designator structure for repeating intervals involves over four hundred lines of regular expression based logic.

```
(defmethod compute-repository-revision ((reference lmdb-repository) (revision-designator
string)
                                        &key  (if-does-not-exist :error)
                                        (class (repository-revision-class reference))
                                        (revision-class class))
  . . .
  (multiple-value-bind (min-designator min-offset max-designator max-offset window-interval
repeat-interval repeat-count)
                       (rlmdb:parse-compound-revision-designator revision-designator)
    . . .
    (make-revision revision-class
      :revision-id (or max-uuid min-uuid) :reference reference
      :min-revision-record min-record :max-revision-record max-record
      :min-revision-ordinal min-ordinal :max-revision-ordinal max-ordinal
      :window-interval window-interval
      :repeat-interval repeat-interval
      :repeat-limit repeat-count
      :designator revision-designator
      :mode modes) )
```

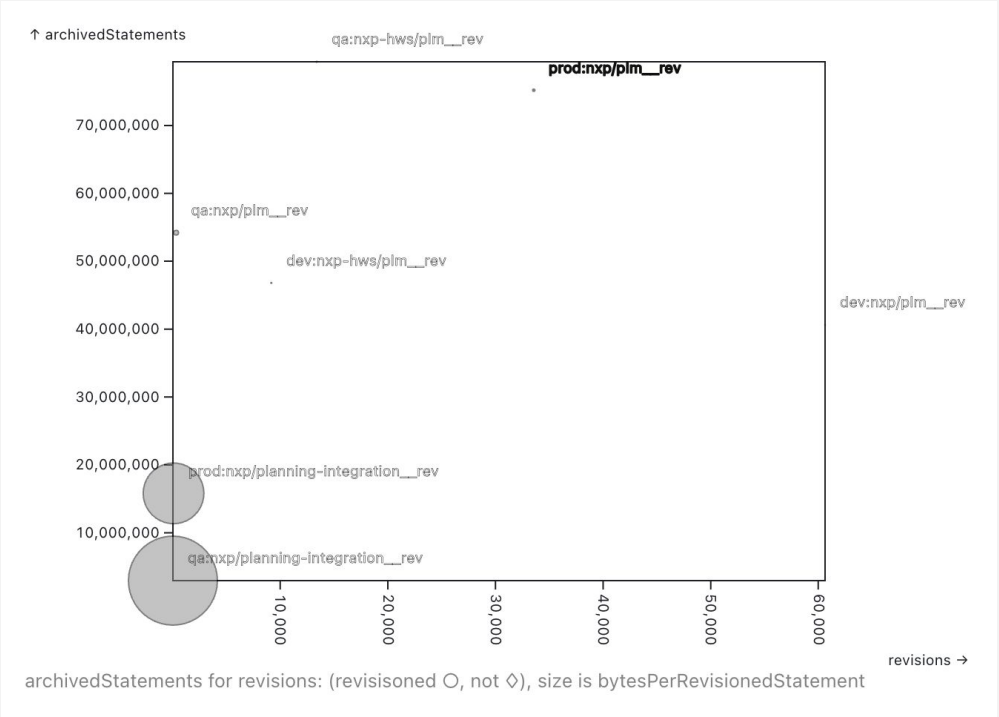
For just an aspect :

```
/^R(\d*)\/(\d{4}(?:-\d{2}){2}T\d{2}(?::\d{2}){2}(?>[^\/]+))\/(P(?=.)(?:\d+Y)?(?:\d+M)?(?:\d+
D)?(?:T(?=.)(?:\d+H)?(?:\d+M)?(?:\d+S)?)?)$/
```

1. intent
2. architecture
3. **space**
   volume
4. access time
5. annotation
6. issues …

Given the approach, how does space relate to usage, how do repository sizes relate to statement and revision count and update patterns? [5]

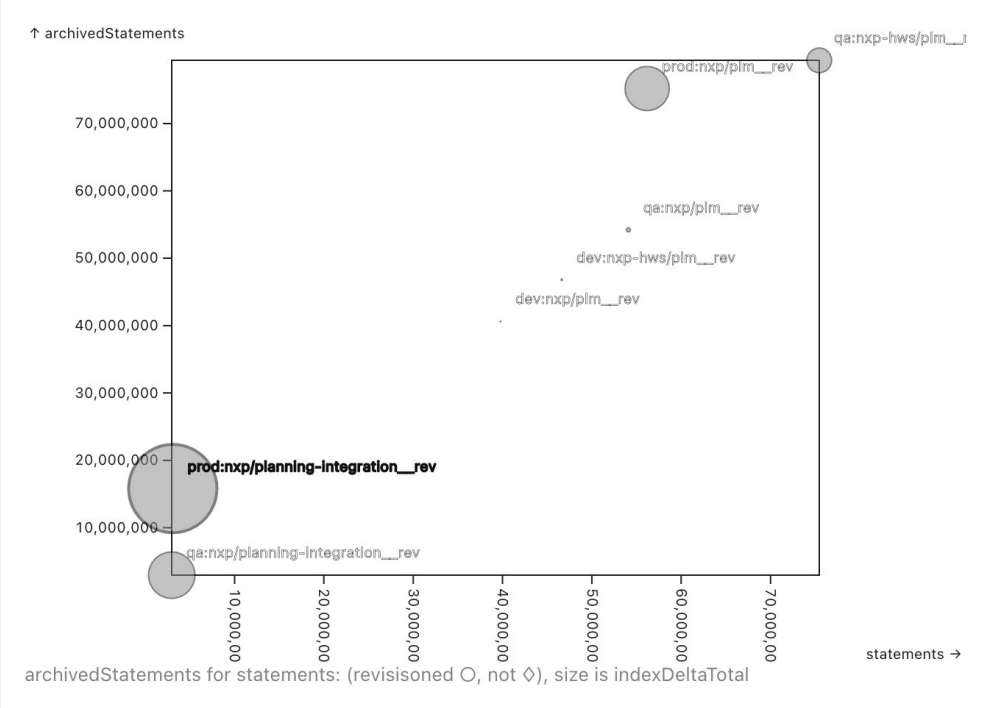
For : archived statements for revisions (bytes per revisioned statement



1. intent
2. architecture
3. **space**
   volume
4. access time
5. annotation
6. issues ...

Given the approach, how does space relate to usage, how do repository sizes relate to statement and revision count and update patterns? [5]

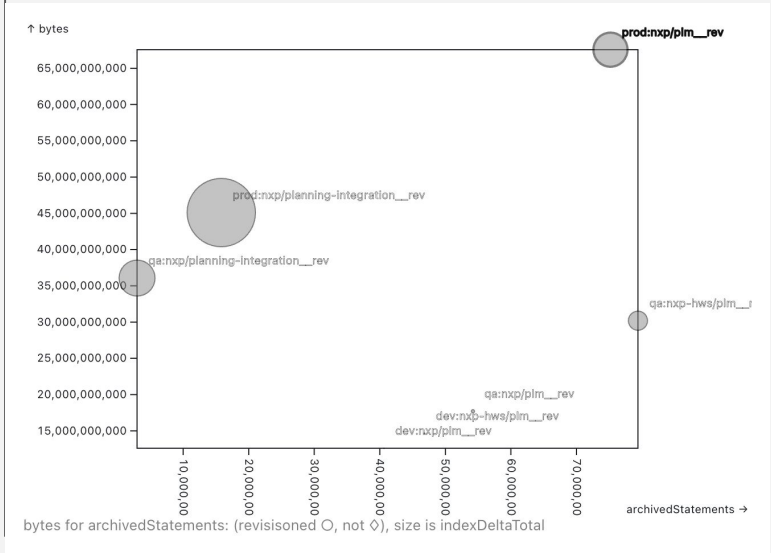
For : archived statements for statements (revision index count)



1. intent
2. architecture
3. **space**
   volume
4. access time
5. annotation
6. issues …

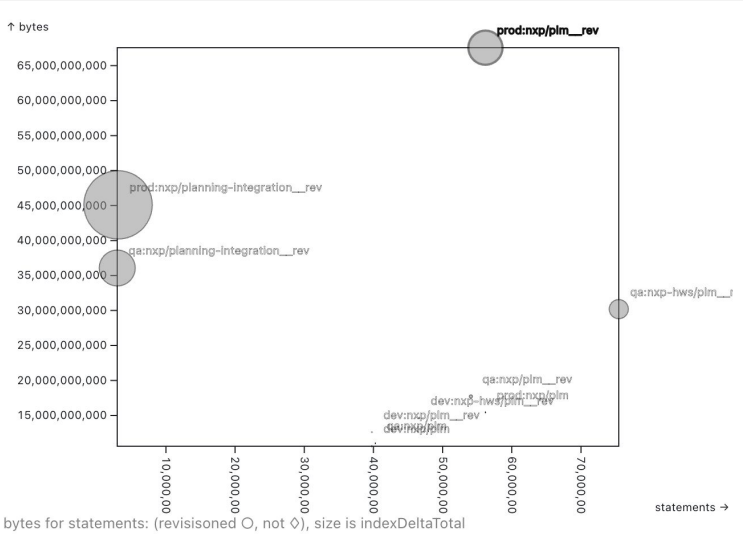
Given the approach, how does space relate to usage, how do repository sizes relate to statement and revision count and update patterns? [5]

For :
bytes for archived statements (index delta)



bytes for archivedStatements: (revisisoned ○, not ◊), size is indexDeltaTotal

For :
bytes for statements (index delta)

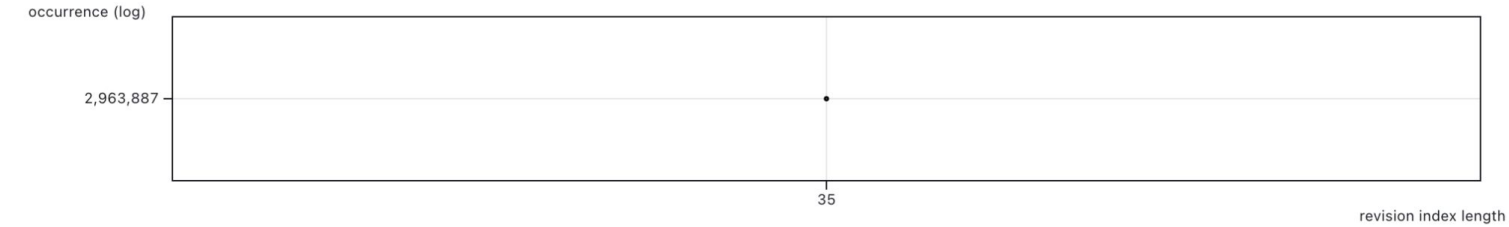


bytes for statements: (revisisoned ○, not ◊), size is indexDeltaTotal

1. intent
2. architecture
3. **space**
   volume
4. access time
5. annotation
6. issues ...

The approach does manifest anomalous update patterns.

For :
anomalous revision index length distribution over time)



Distribution of statement revision index lengths over time
for repository "nl8:nxp/planning-integration__rev"
on 20230716 with 2963887 statements in 19 revisions

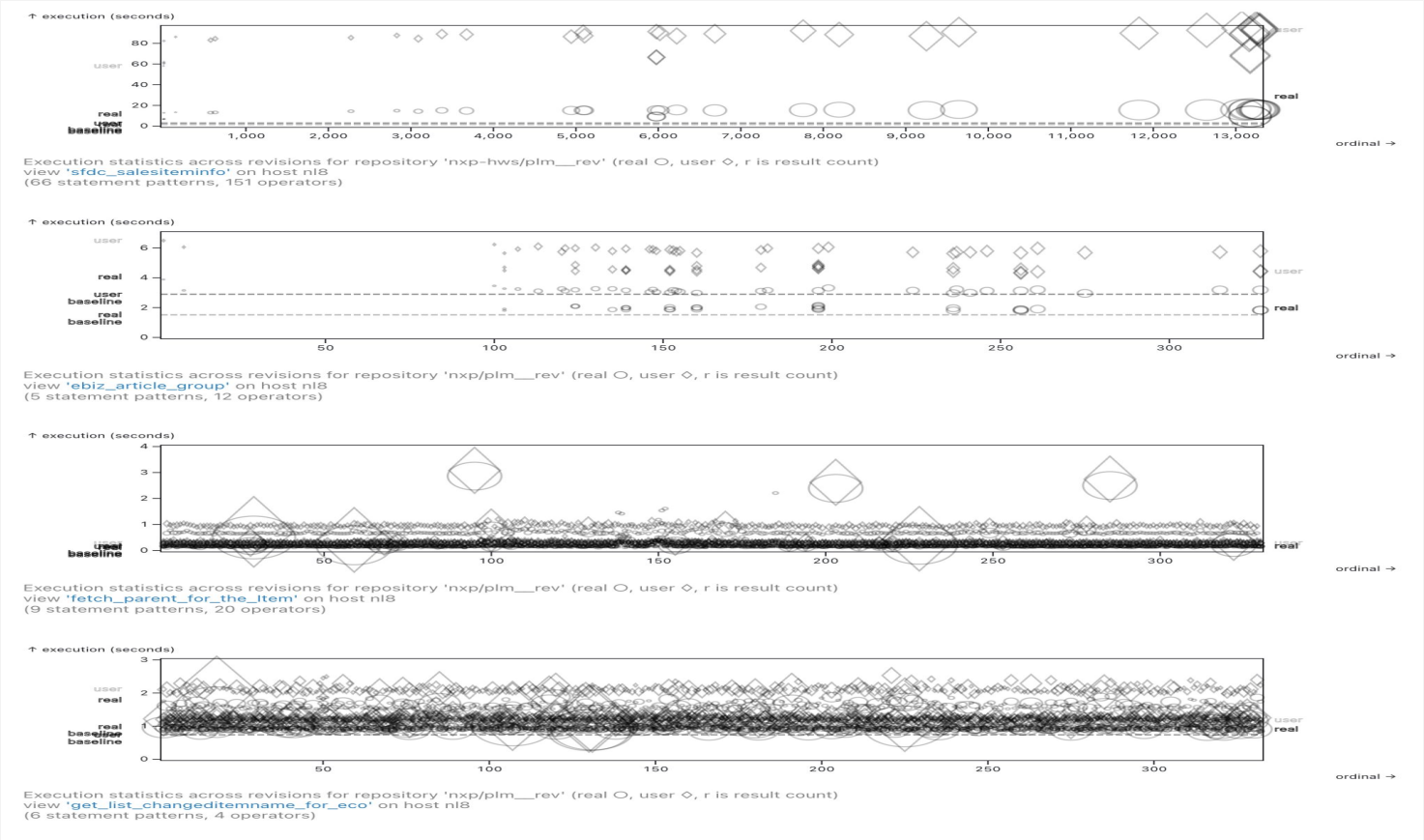
The approach turns out to address the variability issue by limiting the amount of data moved.:

The BGP solution stream generation process contrasts to approaches which materialize the stream of temporally constrained statements - whether explicitly or implicitly, and only then subjects them to a statement pattern. Dydra first applies the pattern to the dataset and then applies temporal constraints. The execution statistics for views reflect this effect: the count of matched statements is a small fraction of the statements which would be included in a materialized version.

view	match count	statement count	percentage
get_list_changeditemname_for_eco	1619419	54085015	2.99
sfdc_salesiteminfo	18627813	75486524	24.68
fetch_parent_for_the_Item	218297	54085015	0.40
ebiz_article_group	3012268	54085015	5.57

1. intent
2. architecture
3. space
4. **access time**
   variability
5. annotation
6. issues ...

The performance benefits are evident where the revision count does not affect query performance.
The graphs, below, depict the execution time of typical queries over the revision history of the respective repositories. [4]

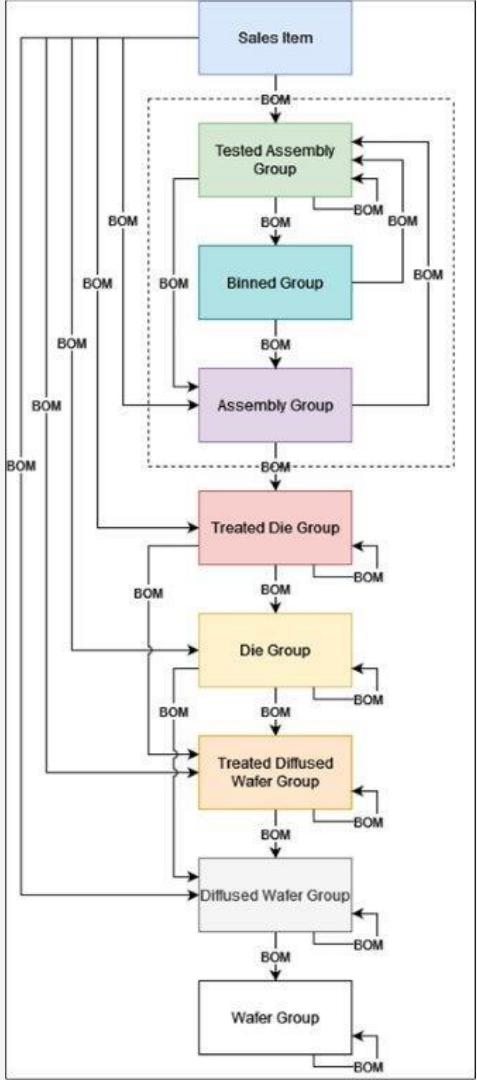


1. intent
2. architecture
3. space
4. access time
5. **annotation**
6. issues ...

Explore and application to a data semiconductor product life cycle model

The typical document is bill-of-materials.

The venue is the vendor collaboration portal. The presentation here depict the effects of change orders on the device characteristics by highlighting the changes in red.



The abstract schema describes each semiconductor device as a "star" relation for the device resource identifier:

```
prefix plm: <http://nxp.com/data> .
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
prefix xsd: <http://www.w3.org/2001/XMLSchema-datatypes> .
plm:Device a rdfs:Class .
plm:name a rdfs:predicate;
    rdfs:domain plm:Device;
    rdfs:range xsd:string .
plm:state  a rdfs:predicate;
    rdfs:domain plm:Device;
    rdfs:range xsd:string .
plm:revision  a rdfs:predicate;
    rdf:domain plm:Device;
    rdfs:range xsd:string .
plm:description  a rdfs:predicate;
    rdfs:domain plm:Device;
    rdfs:range xsd:string .
```

That abstract schema leads to queries built up from "star" patterns.

```
prefix : <urn:dydra:>
prefix plm: <http://example.org/data/>
select *
from :all
where {
  ?s a ?type ;
    plm:name ?name ;
    plm:state ?state ;
    plm:revision ?revision ;
    plm:description ?description .
}
```

In order to indicate the consequences of change orders we introduced, first, properties for Allen interval relations.
For these we extended the SPARQL processor with statement annotations in the manner of RDF Star.

```
prefix : <urn:dydra:>
prefix plm: <http://example.org/data/>
select *
from :all
where {
 ?s a ?type
      {| :met-by ?type_metBy ; :starts ?type_starts |} ;
   plm:name ?name
      {| :met-by ?name_metBy ; :starts ?name_starts |} ;
   plm:state ?state
      {| :met-by ?state_metBy ; :starts ?state_starts |} ;
   plm:revision ?revision
      {| :met-by ?revision_metBy ; :starts ?revision_starts |} ;
   plm:description ?description
      {| :met-by ?description_metBy ; :starts ?description_starts |}  .

 bind(?type_metBy != ?type_starts as ?type_redline)
 bind(?name_metBy != ?name_starts as ?name_redline)
 bind(?state_metBy != ?state_starts as ?state_redline)
 bind(?revision_metBy != ?revision_starts as ?revision_redline)
 bind(?description_metBy != ?description_starts as ?description_redline)
}
```

The limited performance of this approach demonstrated the issue noted above, that revision metadata is not immediately suitable to be included in BGP solutions. A more efficient approach is to reduce interning by applying as much logic as possible in the temporal value domain. Rather than formulating the queries with annotations and filters, a more efficient implementation is to introduce temporal operators which embody the complete logic. In order to circumvent that rate limit, we introduced a virtual temporal attribute to designate an operator which applies the described logic to compute whether the statement had been introduced in a given revision. This attribute was bound in the respective solution for each device parameter to a single boolean variable specific to the parameter.

The result was queries according to the pattern illustrated in figure 17 where the implementation encapsulates this logic in a more optimum system function called not-asserted. This not only executes the above operation more optimally, it also provided for cleaner code. The approach permitted queries of the complexity illustrated in figure 18. The effect of the revised logic was to reduce their execution times from minutes to tens of seconds.

In order to capture indicates of the consequences of change orders we introduced, first, properties for Allen interval relations.
For these we extended the SPARQL processor with statement annotations in the manner of RDF Star.

```
prefix : <urn:dydra:>
prefix plm: <http://example.org/data/>
select *
from :all
where {
  ?s a ?type {| <urn:dydra:notAsserted> ?type_redline |} ;
    plm:name ?name
        {| <urn:dydra:notAsserted> ?name_redline |} ;
    plm:state ?state
        {| <urn:dydra:notAsserted> ?state_redline |} ;
    plm:revision ?revision
        {| <urn:dydra:notAsserted> ?revision_redline|} ;
    plm:description ?description
        {| <urn:dydra:notAsserted>  ?description_redline |}   .
}
```

1. intent
2. architecture
3. space
4. access time
5. **annotation**
6. issues ...

Again, we take things literally.

```
(defgeneric compute-version-lambda (filter-expression annotations)
  (:method (filter-expression annotations)
    (setf filter-expression
          (if filter-expression
              `(handler-case (when ,(if (rest filter-expression) `(and ,@filter-expression) (first
filter-expression))
                               t)
                 (error (c) (when (typep c *break-on-filter-errors*)
                              (break "Error in version predicate: ~a: ~a."
                                     ',filter-expression c))))
              t))
    `(lambda (continuation %quad version-map version-map-length)
       (flet ((continue-for-version (index length min max)
                (declare (ignorable index length min max))
                (let ,(loop for (predicate variable) on annotations by #'cddr
                            collect `(,variable (,predicate version-map index length)))
                  (when ,filter-expression
                    (when *compute-version-lambda.verbose*
                      (print (list :compute-version-lambda.continue
                                   continuation %quad min max
                                   ,@(loop for (nil variable) on annotations by #'cddr collect
variable))))
                    (funcall continuation %quad min max
                             ,@(loop for (nil variable) on annotations by #'cddr collect variable))))))
         (declare (dynamic-extent #'continue-for-version))
         (apply #'map-statement-versions version-map version-map-length #'continue-for-version
                (transaction-version-constraints *transaction*))))))
```

1. intent
2. architecture
3. space
4. access time
5. **annotation**
6. issues ...

Again, for the implementation of how to interpret a statement pattern, we take things literally.

```
(let* (. . .
       (quad-pattern (destructuring-bind (g s p o) concrete-match-pattern `(vector ,g ,s ,p ,o)))
       (version-lambda (when version-annotation (compute-version-lambda version-filter-expression
version-annotation))))
  `(labels ((,lmdb-continuation (%quad &optional first last ,@version-variables)
     (rlmdb::map-repository-statements-filtered #',lmdb-continuation transaction
                                                ,quad-pattern
                                                #'version-operator))
   . . .
```

Again, for the implementation of how to interpret a statement pattern, we take things literally.

```
((or wild-pattern-p (%quad-match-p %quad-pattern %index-quad)) ;; iff still in range
                                          (cond ((zerop visibility-count) ; not revisioned, should not
be here
                                                 (map-repository-statements-filtered-callback operator

(term-number-key-to-term-number-quad %index-quad %result-quad quad-map)
                                                                                                  nil nil)
                                                  (incf match-count))
                                                (t
                                                 ;; maps the quad even if the filter fails
                                                 (incf match-count (funcall filter operator

(term-number-key-to-term-number-quad %index-quad %result-quad quad-map)
                                                                                %index-map
visibility-count)))))
```

The result was queries according to the pattern illustrated below where the implementation encapsulates this logic in a more optimum system function called not-asserted. This not only executes the above operation more optimally, it also provided for cleaner code. The effect of the revised logic was to reduce their execution times from minutes to tens of seconds.

```
SELECT DISTINCT ?VAR2 ?VAR41 ?VAR12 ?VAR1 ?VAR4 ?VAR18 ?VAR21 ?VAR24 ?VAR45 ?VAR7 ?VAR15 ?VAR27 ?VAR30 ?VAR33 ?VAR36 ?VAR39 ?VAR13 ?VAR5 ?VAR19 ?VAR22 ?VAR25
?VAR8 ?VAR16 ?VAR28 ?VAR31 ?VAR34 ?VAR40 ?VAR37 ?VAR201 ?VAR172 ?VAR174 ?VAR178 ?VAR170 ?VAR173 ?VAR175 ?VAR179 ?VAR61 ?VAR64 ?VAR67 ?VAR70 ?VAR73 ?VAR76
?VAR79 ?VAR82 ?VAR85 ?VAR87 ?VAR92 ?VAR59 ?VAR62 ?VAR65 ?VAR68 ?VAR71 ?VAR74 ?VAR77 ?VAR80 ?VAR83 ?VAR86 ?VAR88 ?VAR93 ?VAR102 ?VAR105 ?VAR108 ?VAR111
?VAR114 ?VAR118 ?VAR99 ?VAR103 ?VAR106 ?VAR109 ?VAR112 ?VAR115 ?VAR119 ?VAR126 ?VAR129 ?VAR131 ?VAR133 ?VAR136 ?VAR139 ?VAR142 ?VAR145 ?VAR148 ?VAR151
?VAR158 ?VAR122 ?VAR155 ?VAR161 ?VAR164 ?VAR167 ?VAR127 ?VAR130 ?VAR132 ?VAR134 ?VAR137 ?VAR140 ?VAR143 ?VAR146 ?VAR149 ?VAR152 ?VAR159 ?VAR153 ?VAR156
?VAR162 ?VAR165 ?VAR168 ?VAR47 ?VAR53 ?VAR55 ?VAR52 ?VAR48 ?VAR54 ?VAR56 ?VAR181 ?VAR191 ?VAR193 ?VAR196 ?VAR199 ?VAR190 ?VAR182 ?VAR192 ?VAR194 ?VAR197
?VAR200
 WHERE { { { { { SERVICE <http://example.org/0>
                          { ?type <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2000/01/rdf-schema#Class> .
                            ?type <http://www.w3.org/2000/01/rdf-schema#label> ?VAR1 }
                  { { { { { { { { { ?VAR2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type .
                                    ?VAR2 <http://example.org/3> ?VAR4 {|<urn:dydra:notAsserted> ?VAR5|} .
                                    ?VAR2 <http://example.org/6> ?VAR7 {|<urn:dydra:notAsserted> ?VAR8|} .
                                    ?VAR2 <http://example.org/9> ?VAR10 .
                                    ?VAR2 <http://example.org/11> ?VAR12 {|<urn:dydra:notAsserted> ?VAR13|} }
                                  OPTIONAL
                                  { ?VAR2 <http://example.org/14> ?VAR15 {|<urn:dydra:notAsserted> ?VAR16|} } }}
                                OPTIONAL
                                { ?VAR2 <http://example.org/17> ?VAR18 {|<urn:dydra:notAsserted> ?VAR19|} } }}
                              OPTIONAL
                              { ?VAR2 <http://example.org/20> ?VAR21 {|<urn:dydra:notAsserted> ?VAR22|} } }}
                            OPTIONAL
                            { ?VAR2 <http://example.org/23> ?VAR24 {|<urn:dydra:notAsserted> ?VAR25|} } }}
                          OPTIONAL
                          { ?VAR2 <http://example.org/26> ?VAR27 {|<urn:dydra:notAsserted> ?VAR28|} } }}
                        OPTIONAL
                        { ?VAR2 <http://example.org/29> ?VAR30 {|<urn:dydra:notAsserted> ?VAR31|} } }}
                      OPTIONAL
                      { ?VAR2 <http://example.org/32> ?VAR33 {|<urn:dydra:notAsserted> ?VAR34|} } }}
                    OPTIONAL
                    { ?VAR2 <http://example.org/35> ?VAR36 {|<urn:dydra:notAsserted> ?VAR37|} } }}
                  OPTIONAL
                  { ?VAR2 <http://example.org/38> ?VAR39 {|<urn:dydra:notAsserted> ?VAR40|} } }}
                BIND (( if((?VAR12 = 'Yes'), '', ( if((?VAR12 = 'No') && ( strstarts(?VAR10, 'CP-ATX-')), 'L-FSL', 'L-NXP'))))) AS?VAR41) }
```

The result was queries according to the pattern illustrated below where the implementation encapsulates this logic in a more optimum system function called not-asserted. This not only executes the above operation more optimally, it also provided for cleaner code. The effect of the revised logic was to reduce their execution times from minutes to tens of seconds.

```
...
                  BIND (( if((?VAR12 = 'Yes'), '', ( if((?VAR12 = 'No') && ( strstarts(?VAR10, 'CP-ATX-')), 'L-FSL', 'L-NXP'))))) AS?VAR41) }
                OPTIONAL
                SELECT ?VAR2 ?VAR45
                 WHERE { ?VAR2
(^<http://example.org/42>+/<http://example.org/43>/<http://example.org/43>/<http://example.org/43>/<http://example.org/43>/<http://example.org/43>/<http://ex
ample.org/43>/<http://example.org/44>) ?VAR45 } }}
              OPTIONAL
              { { { { { { ?VAR46 <http://example.org/3> ?VAR47 {|<urn:dydra:notAsserted> ?VAR48|} .
                          ?VAR46 <http://example.org/6> ?VAR49 }
                        { {  { ?VAR46 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/50> }
                           UNION { ?VAR46 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/51> }}
                          { ?VAR2 <http://example.org/42> ?VAR46 {|<urn:dydra:notAsserted> ?VAR52|} } } }
                     OPTIONAL
                     { ?VAR46 <http://example.org/17> ?VAR53 {|<urn:dydra:notAsserted> ?VAR54|} } }}
                   OPTIONAL
                   { ?VAR46 <http://example.org/14> ?VAR55 {|<urn:dydra:notAsserted> ?VAR56|} } }}
                 OPTIONAL
                 { { { { { { { { { { ?VAR46 <http://example.org/57> ?VAR58 {|<urn:dydra:notAsserted> ?VAR59|} .
                                    ?VAR58 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/60> .
                                    ?VAR58 <http://example.org/3> ?VAR61 {|<urn:dydra:notAsserted> ?VAR62|} .
                                    ?VAR58 <http://example.org/6> ?VAR63 }
                                  OPTIONAL
                                  { ?VAR58 <http://example.org/17> ?VAR64 {|<urn:dydra:notAsserted> ?VAR65|} } }}
                                OPTIONAL
                                { ?VAR58 <http://example.org/66> ?VAR67 {|<urn:dydra:notAsserted> ?VAR68|} } }}
                              OPTIONAL
                              { ?VAR58 <http://example.org/69> ?VAR70 {|<urn:dydra:notAsserted> ?VAR71|} } }}
                            OPTIONAL
                            { ?VAR58 <http://example.org/72> ?VAR73 {|<urn:dydra:notAsserted> ?VAR74|} } }}
                          OPTIONAL
                          { ?VAR58 <http://example.org/75> ?VAR76 {|<urn:dydra:notAsserted> ?VAR77|} } }}
                        OPTIONAL
                        { ?VAR58 <http://example.org/78> ?VAR79 {|<urn:dydra:notAsserted> ?VAR80|} } }}
```

The result was queries according to the pattern illustrated below where the implementation encapsulates this logic in a more optimum system function called not-asserted. This not only executes the above operation more optimally, it also provided for cleaner code. The effect of the revised logic was to reduce their execution times from minutes to tens of seconds.

...

```
                    OPTIONAL
                    { ?VAR58 <http://example.org/81> ?VAR82 {|<urn:dydra:notAsserted> ?VAR83|} } }}
                  OPTIONAL
                  { ?VAR58 <http://example.org/84> ?VAR85 {|<urn:dydra:notAsserted> ?VAR86|} } }}
                OPTIONAL
                { ?VAR58 <http://example.org/14> ?VAR87 {|<urn:dydra:notAsserted> ?VAR88|} } }}
              OPTIONAL
              SELECT ?VAR58 (( max(( bound(?VAR90)))) as ?VAR92) (( max(?VAR91)) as ?VAR93)
               WHERE { ?VAR58 <http://example.org/89> ?VAR90 {|<urn:dydra:notAsserted> ?VAR91|} }
              GROUP BY  ?VAR58 }} }
            FILTER not in (?VAR63, ('OBS', 'DEV'))}
          OPTIONAL
          { { { { { { SERVICE <http://example.org/94>
                              { { ?VAR95 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/96> .
                                  ?VAR95 <http://www.w3.org/2000/01/rdf-schema#label> ?VAR97 } FILTER
                                not in (?VAR97, ('OBS', 'DEV')) }
                      { ?VAR46 <http://example.org/57> ?VAR98 {|<urn:dydra:notAsserted> ?VAR99|} .
                        ?VAR98 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/100> .
                        ?VAR98 <http://example.org/101> ?VAR102 {|<urn:dydra:notAsserted> ?VAR103|} .
                        ?VAR98 <http://example.org/104> ?VAR105 {|<urn:dydra:notAsserted> ?VAR106|} .
                        ?VAR98 <http://example.org/96> ?VAR95 } }
                    OPTIONAL
                    { ?VAR98 <http://example.org/107> ?VAR108 {|<urn:dydra:notAsserted> ?VAR109|} } }}
                  OPTIONAL
                  { ?VAR98 <http://example.org/110> ?VAR111 {|<urn:dydra:notAsserted> ?VAR112|} } }}
                OPTIONAL
                { ?VAR98 <http://example.org/113> ?VAR114 {|<urn:dydra:notAsserted> ?VAR115|} } }}
              OPTIONAL
              { SERVICE <http://example.org/94>
                         { ?VAR116 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/117> .
                           ?VAR116 <http://www.w3.org/2000/01/rdf-schema#label> ?VAR118 }
                { ?VAR98 <http://example.org/117> ?VAR116 {|<urn:dydra:notAsserted> ?VAR119|} } } }}
            OPTIONAL
```

The result was queries according to the pattern illustrated below where the implementation encapsulates this logic in a more optimum system function called not-asserted. This not only executes the above operation more optimally, it also provided for cleaner code. The effect of the revised logic was to reduce their execution times from minutes to tens of seconds.

```
...
                { SERVICE <http://example.org/94>
                          { ?VAR120 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/121> .
                            ?VAR120 <http://www.w3.org/2000/01/rdf-schema#label> ?VAR122 }
                  { { { { { { SERVICE <http://example.org/94>
                                      { { ?VAR123 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/96> .
                                          ?VAR123 <http://www.w3.org/2000/01/rdf-schema#label> ?VAR124 } FILTER
                                        not in (?VAR124, ('OBS', 'DEV')) }
                                    { ?VAR98 <http://example.org/125> ?VAR126 {|<urn:dydra:notAsserted> ?VAR127|} .
                                      ?VAR126 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/128> .
                                      ?VAR126 <http://example.org/101> ?VAR129 {|<urn:dydra:notAsserted> ?VAR130|} .
                                      ?VAR126 <http://example.org/104> ?VAR131 {|<urn:dydra:notAsserted> ?VAR132|} .
                                      ?VAR126 <http://example.org/107> ?VAR133 {|<urn:dydra:notAsserted> ?VAR134|} .
                                      ?VAR126 <http://example.org/135> ?VAR136 {|<urn:dydra:notAsserted> ?VAR137|} .
                                      ?VAR126 <http://example.org/138> ?VAR139 {|<urn:dydra:notAsserted> ?VAR140|} .
                                      ?VAR126 <http://example.org/141> ?VAR142 {|<urn:dydra:notAsserted> ?VAR143|} .
                                      ?VAR126 <http://example.org/144> ?VAR145 {|<urn:dydra:notAsserted> ?VAR146|} .
                                      ?VAR126 <http://example.org/147> ?VAR148 {|<urn:dydra:notAsserted> ?VAR149|} .
                                      ?VAR126 <http://example.org/150> ?VAR151 {|<urn:dydra:notAsserted> ?VAR152|} .
                                      ?VAR126 <http://example.org/121> ?VAR120 {|<urn:dydra:notAsserted> ?VAR153|} .
                                      ?VAR126 <http://example.org/96> ?VAR123 } }
                                  OPTIONAL
                                  { ?VAR126 <http://example.org/154> ?VAR155 {|<urn:dydra:notAsserted> ?VAR156|} } }}
                                OPTIONAL
                                { ?VAR126 <http://example.org/157> ?VAR158 {|<urn:dydra:notAsserted> ?VAR159|} } }}
                              OPTIONAL
                              { ?VAR126 <http://example.org/160> ?VAR161 {|<urn:dydra:notAsserted> ?VAR162|} } }}
                            OPTIONAL
                            { ?VAR126 <http://example.org/163> ?VAR164 {|<urn:dydra:notAsserted> ?VAR165|} } }}
                          OPTIONAL
                          { ?VAR126 <http://example.org/166> ?VAR167 {|<urn:dydra:notAsserted> ?VAR168|} } }} } }} }
            FILTER not in (?VAR49, ('OBS', 'DEV'))}
          OPTIONAL
          { { ?VAR2 <http://example.org/57> ?VAR169 {|<urn:dydra:notAsserted> ?VAR170|} .
```

The result was queries according to the pattern illustrated below where the implementation encapsulates this logic in a more optimum system function called not-asserted. This not only executes the above operation more optimally, it also provided for cleaner code. The effect of the revised logic was to reduce their execution times from minutes to tens of seconds.

```
...
            FILTER not in (?VAR49, ('OBS', 'DEV'))}
          OPTIONAL
          { { ?VAR2 <http://example.org/57> ?VAR169 {|<urn:dydra:notAsserted> ?VAR170|} .
              ?VAR169 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/171> .
              ?VAR169 <http://example.org/3> ?VAR172 {|<urn:dydra:notAsserted> ?VAR173|} .
              ?VAR169 <http://example.org/17> ?VAR174 {|<urn:dydra:notAsserted> ?VAR175|} }
            OPTIONAL
            SELECT ?VAR169 (( max(( bound(?VAR176)))) as ?VAR178) (( max(?VAR177)) as ?VAR179)
             WHERE { ?VAR169 <http://example.org/89> ?VAR176 {|<urn:dydra:notAsserted> ?VAR177|} }
            GROUP BY  ?VAR169 }} }}
        OPTIONAL
        { { { { { { ?VAR180 <http://example.org/3> ?VAR181 {|<urn:dydra:notAsserted> ?VAR182|} .
                    ?VAR180 <http://example.org/6> ?VAR183 }
                  { {  { ?VAR180 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/184> }
                       UNION { ?VAR180 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/185> }}
                    SELECT (?VAR189 as ?VAR180) ?VAR2 (( max(?VAR188)) as ?VAR190)
                     WHERE { { SELECT ?VAR187 ?VAR186 ?VAR2 ?VAR188
                                WHERE { ?VAR2 <http://example.org/42>* ?VAR186 .
                                        ?VAR186 <http://example.org/42> ?VAR187 {|<urn:dydra:notAsserted> ?VAR188|} }
                               SELECT ?VAR189 ?VAR187 ?VAR2
                                WHERE { ?VAR2 <http://example.org/42>+ ?VAR189 .
                                        ?VAR187 <http://example.org/42>* ?VAR189 } }}
                    GROUP BY  ?VAR189 ?VAR2 } }
                OPTIONAL
                { ?VAR180 <http://example.org/17> ?VAR191 {|<urn:dydra:notAsserted> ?VAR192|} } }}
              OPTIONAL
              { ?VAR180 <http://example.org/14> ?VAR193 {|<urn:dydra:notAsserted> ?VAR194|} } }}
            OPTIONAL
            { ?VAR180 <http://example.org/195> ?VAR196 {|<urn:dydra:notAsserted> ?VAR197|} } }}
          OPTIONAL
          { ?VAR180 <http://example.org/198> ?VAR199 {|<urn:dydra:notAsserted> ?VAR200|} } }} }
        FILTER not in (?VAR183, ('OBS', 'DEV'))}}
```

1. intent
2. architecture
3. import time
4. import space
5. query time
6. **issues** ...

This experience raises several issues :

1. How do the observed space requirements for TB storage based in visibility vectors relate to that of CB storage?
2. Which update patterns are support by this storage model - the counter-example being PUT updates?
3. How best to accommodate the term-id domain v/s temporal value domain disjunction. Is it possible to handle that with a smart compiler rather than requiring native operators?
4. Would limits to the number of unique revision index variants permit a dictionary to reduce space requirements for visibility records?

[1] : Laney, Doug. "3D data management: Controlling data volume, velocity and variety." *META group research note* 6, no. 70 (2001): 1. (2012)
[2] : Anderson, James, and Arto Bendiken. "Transaction-Time Queries in Dydra." *MEPDaW/LDQ@ ESWC* 1585 (2016): 11-19.
[3] : Anderson, James. "RDF graph stores as convergent datatypes." In *Companion Proceedings of The 2019 World Wide Web Conference*, pp. 940-942. 2019.
[4] : Datagraph GmbH : https://observablehq.com/@datagenous/revisioned-repository-execution-statistics
[5] : Datagraph GmbH : https://observablehq.com/@datagenous/revisioned-repository-size-statistics