Access Control on Big Data and Small Pixels: How to Achieve Privacy and Security

Peter Baumann, Dimitar Mišev
Jacobs University, p.baumann@jacobs-university.de, d.misev@jacobs-university.de

***Abstract* –** With high-volume, function-rich Web services on spatio-temporal raster data on the rise there is a growing need for adaptive access control. However, while the datacube paradigm has proven suitable for large-scale, efficient raster services it is not yet clear how appropriate access control mechanisms can be provided to service administrators. In this contribution we provide a model which extends the SQL datacube query language, SQL/MDA, with flexible definition and enforcement of array-aware access control.

Index Terms – access control, array, coverage, datacube, rasdaman

MOTIVATION

Web services offering scientific data are emerging in increasing numbers, based on the progress technology has made recently. There is a strong movement towards open data. A radical interpretation of open data is that (observed or generated) data should be available to everybody without restriction and pricing. However, there are more differentiated views, too. For industrial and safety or privacy critical governmental services it is obvious that access has to be controlled tightly. However, even scientists acting in non-commercial and often not safety-critical environments (here a maximum of free information flow is theoretically possible) make strong claims for protecting access to data: researchers enjoy protection of their data for some grace period to grant them a – generally considered legitimate – right to publish any findings first.

This considers data sets as a whole, i.e., a "all or nothing" access. However, with multi-Petabytes getting online [1] this is no longer adequate; rather, access protection also of spatial and/or temporal regions must be considered. For example, the European Centre for Medium-Range Weather Forecast (ECMWF) offers very long timeseries of climate data. While the long tail of data is accessible for free the most recent two weeks are priced.

Further, services tend to evolve from pure data extraction and download to more and more flexible, user-specified processing in the server prior to downloading results. Given that power of executing foreign code in a server the processing cycles themselves become a cost-relevant factor which, however, is highly fluctuating between requests and, therefore, hard to oversee for the service provider as compared to fixed-effort requests.

In this paper we focus on spatio-temporal datacubes [2], our research question being: *how can effective access control be achieved on massive datacubes?* Obviously, such a mechanism should be aware of the characteristics of datacubes in both data and service aspects. For example, access control must be able to differentiate access to different regions and times and react individually.

Our research – manifest in the rasdaman scalable, distributed datacube engine [3] – specifically concentrates on datacube services based on a declarative array query language (which meantime has been incorporated in the ISO SQL standard [4]). Hence, we propose an array access control mechanism based on the following key requirements:

- Based on the proven standards of Role-Based Access Control (RBAC) as per SQL;
- Array specific access specification and granularity, in particular: protecting arbitrary regions in an n-D array, down to the level of single pixels;
- Allow mandatory access control;
- Amenable to the various query optimization techniques.

In our approach we use SQL triggers for defining constraints on array access and processing, together with actions executed whenever a constraint is violated. Such triggers are always applied to queries, except when the user firing the query has been exempted by the administrator. Array-specific constraints can rely on array query expressions, which are suitably enhanced with new operators.

In the sequel, we consider the state of the art, introduce array-aware access control, provide representative use cases, and give conclusions.

STATE OF THE ART

SQL has outstandingly elaborate access control functionality [5]. User privilege management in standard databases is done through Role-Based Access Control (RBAC). Clients open database sessions under some known user name, at the same time undergoing authentication; this determines the privileges the session will have, one of SELECT, INSERT, UPDATE, and DELETE; role hierarchies help to organize such privileges for the various users. There is no specific support for array access control in SQL.

The MongoDB NoSQL system offers privileges listing allowed operations, but no fine-grain access into its objects.

The SciDB Array DBMS defines access control only on complete arrays, actually: so-called namespaces which are directory-like collections of arrays [9]. This is too inflexible

and coarse-grain for the advanced use cases under discussion here.

As SQL until recently did not support datacubes, such services (including satellite imagery and climate / weather archives) resorted to ad-hoc implementations which usually are not aware of many of central database features such as declarative query languages and fine-grain access control. Classically, therefore, satellite imagery as well as weather and climate data centers operate on semantic-less files as atomic units of access. Consequently, data type specific access control is not possible, including safeguarding, for example, arbitrary regions in a multi-dimensional datacube. In the widely used OPeNDAP Hyrax server, for example, access control is a known open issue [10].

A series of systems, such as the Australian Geoscience Data Cube [8], offer access via python programming. Client authentication is forwarded to the operating system, meaning that operating system login names are exposed to the Internet. Access to objects is done on a file basis, so again authorization is delegated to the operating system. Array-specific authorization (such as protecting particular regions) is not supported.

A principal problem is posed by the expressiveness of a service: the more powerful it is, the harder it is to control. For example, allowing any client out in wilderness to ship any foreign python code to a server and execute it there without further precaution establishes a first-class goal for all sorts of attacks – it is well known in Computer Science that a program can never comprehend other procedural (i.e., Turing-complete) programs in general, so there is no way of sufficient automatic protection. Hence, procedural code is too dangerous for service APIs. Better in this respect are declarative languages – like SQL – which tentatively are restricted in their power to a degree that such automatic checking is possible; additionally, this opens the door for all sorts of automatic optimization of queries prior to evaluation, including parallelization and distribution.

This is where SQL triggers [5] come into play. Triggers represent one way of specifying (possibly multi-table) constraints on high level. A trigger basically consists of an event (such as an INSERT or UPDATE operation), a condition to be evaluated whenever that event occurs, and an action to be executed should the constraint be fulfilled. Introduced mainly for automated database consistency control, triggers form a proven tool: definition is high-level, they can be adjusted anytime without programming, and are amenable to all query optimization mechanisms. However, they likewise do not support fine-grain array access control.

In conclusion, there is no mechanism currently available for array-friendly access control. SQL has the most convenient concepts with RBAC and triggers, but does not support arrays either.

ARRAY-AWARE ACCESS CONTROL

In this section we introduce a concept which makes SQL RBAC array-aware. We rely on the standard user / role paradigm as hooks for all database privileges, plus SQL triggers.

I. Triggers

First, we extend the classical trigger concept by adding SELECT to the list of allowed events so as to guard read access, too (ISO SQL only allows INSERT, UPDATE; and DELETE). In the condition specification of a trigger we naturally allow array SQL expressions. This allows expressing data specific conditions, but not yet access. To this end we introduce two predicates, ACCESSED(A) and MODIFIED(A) where *A* is some array-valued expression. The result is a Boolean array of the same size as A where each cell contains *true* iff this cell in *A* gets accessed by the query under consideration, or modified, respectively. By collapsing this Boolean array through an aggregation regions in the array can be specified conveniently, such as

MDANY(ACCESSED(A[100:200,300:400]))

This checks whether any of the cells in the interval between corner points (100,300) and (200,400) undergoes a read access.

For the case that a condition is fulfilled a corresponding action is defined with the trigger, such as aborting the query:

EXCEPTION “Error: insufficient privileges on object.”

Next, we connect triggers to users and roles. We remember that normally a trigger is always checked, so we do not need to add a trigger to user or role privileges; rather, we need to remove – i.e.: deactivate – it. To accomplish this we enhance the SQL GRANT statement with an option to exempt a user or role from a particular trigger:

GRANT EXEMPTION FROM TRIGGER t TO role

The semantics is that from now on all queries executed under this role will not involve activation of the trigger – hence, its corresponding restriction is waived. This actually yields a safe model for restrictions: in the first place, constraints always hold; they need to be explicitly and consciously disabled.

II. Context

So far we have addressed unconditional access restrictions for particular users. Sometimes, however, this should be more dynamic. Quota, for example, should fire only in case some assigned threshold is exceeded.

In our approach we still rely on the privilege/trigger concept, but enhance it with knowledge about the current query through an addressable context object. Among others, this object provides information about data accessed, query result volume, and compute resources spent. In the next section we show how this can be used to define various quota policies.

As triggers apply prior to query execution some of the above context items are not yet available. Such information is provided through the cost-based query optimizer which anyway needs to calculate query costs in order to find a part-

icularly efficient evaluation strategy. Such strategy takes into account the actual data situation, such as object locations, array tiling, and sizing parameters. Decision criteria include: most efficient formulation of expressions; parallelization potential, including use of heterogeneous hardware; distributed processing in federations. For the avoidance of doubts, this of course is an a-priori estimate which very much depends on the accuracy achieved. Obtaining accurate estimates is an area of active research in our array query optimization work, to be published in a forthcoming paper.

If allowed by policy such estimates can be made available to a client in advance so that the client still can decide whether it wants to fire off a particular heavy-weight query.

Finally, billing records can be generated a posteriori, based on the actual resource consumption (which in turn allows an assessment of the optimizer's estimate quality).

USE CASES

In this Section we present practical use cases for defining access control policies using the "trigger as privilege" concept. Citing our initial example we assume a 4-D climate data collection *ERA5* holding some amount of global climate variables over some period of time.

I. Protecting time slices

In the introductory example ECMWF protects its most recent two weeks of data. Below the most recent data are characterized by sub-datacube *S*:

```
CREATE TRIGGER Latest_2_weeks_disallowed
 SELECT ON ERA5
 WHEN
  MDANY( ACCESSED(  ERA5[S] ) )
 BEGIN
  EXCEPTION "Error: no access rights on this area."
 END
```

The logins of governmental organizations, for example, could be exempted from this trigger to get free access. Other users might get access, but with a record added to their bill.

On a side note, as this query obviously fails even if only one single disallowed pixel is accessed: without difficulty this predicate can be adjusted for an overlap by at least N pixels, X percent of the forbidden area, or some other empirical measure.

II. Protecting areas

Assume *P* is a vector polygon over some area, expressed in WKT (Well-Known Text). Then, the following query will disallow access to the *P* area:

```
CREATE TRIGGER Protect_by_Area
 SELECT ON ERA5
 WHEN
   MDANY( ACCESSED(  clip( ERA5, P ) ) )
 BEGIN
  EXCEPTION "Error: no access rights on this area."
 END
```

Note that clip(), while available in rasdaman, is not yet standardized in SQL/MDA, but foreseen as future work.

III. Protecting by mask

Areas accessible can be described in any possible way, including involving another object – of course, access must be allowed for the query to this additional object as well, so this requires careful policy design. Below we use a mask object *ERAmask* given by a Boolean array of the same size as the primary array. Should a cell get accessed where the corresponding mask cell value is *true* then access gets denied:

```
CREATE TRIGGER Protect_by_Mask
 SELECT ON ERA5, ERAmask
 WHEN
  MDANY( ACCESSED( ERA5 ) AND ERAmask )
 BEGIN
  EXCEPTION "Error: no access rights on this area."
 END
```

IV. Quota

Various quota situations can be modeled through triggers. First, we assume a user is constrained by the amount of data accessed, e.g., 1 MB as below:

```
CREATE TRIGGER Quota_on_Access
 SELECT ON ERA5
 WHEN
  MDCOUNT_TRUE( ACCESSED(  ERA5  ) ) > 1000000
 BEGIN
  EXCEPTION "Error: data access volume exceeded."
 END
```

Dynamic quota can be defined in conjunction with some appropriate database modeling where past consumption is recorded. In this case, the trigger can extract the remaining query budget rather than using the constant value shown in the above example.

During our many years of experience with massive queryable datacubes we have learnt that for users it is often not easy to oversee how many data they actually download – the new quality of direct access and manipulation, paired with fast processing, frequently makes them over-enthusiastic about network bandwidth. Earlier download quota, however, required manual programming and still were a-posteriori – it could happen that massive data were produced on the server just to be prevented from downloading them in the end. Now guarding downloads can be easily expressed by the administrator and checked a-priori:

```
CREATE TRIGGER Quota_on_Download
 SELECT ON ERA5
 WHEN
  CONTEXT.COST.RESULTVOLUME > 1000000000
 BEGIN
  EXCEPTION "Error: download volume exceeded."
 END
```

Finally, federations can be protected from excessive load generation. The following trigger prohibits federation

queries entirely, effectively restricting queries to local access:

```
CREATE TRIGGER Disable_Federation
WHEN
  CONTEXT.COST.TRANSFERVOLUME > 0
BEGIN
  EXCEPTION
    "Error: Federated processing required for answering this
    query, but rights for this user/roles are insufficient."
END
```

Note that there is no constraint on operation nor object in this trigger, so it will always apply. Note also that the execution plan assessed is always an efficient one generated by the optimizer, and therefore the query will execute locally if possible – only if federated processing is inevitable a query thus guarded will fail.

CONCLUSION

With the advent of even more "Big Data" offered online in combination with more powerful and flexible services on them we need to consider access control for reasons of data privacy and security, service attack protection, general housekeeping such as quota, and more. Generic mechanisms like file-based access control and role-based access control reach their limits when it comes to complex conditions which require knowledge of the particular data model, such as multi-dimensional arrays.

We propose the "trigger as privilege" concept as a natural enhancement of the proven RBAC model available with standard SQL systems. Assuming an array model and its operations present, such as in the ISO SQL/MDA standard, we introduce

- A trivial extension to triggers to also observe SELECTs;
- Array-specific predicates providing meta-information about the query wrt. data locations read or updated;
- An extension of the GRANT (and REVOKE) statement to explicitly exempt particular users and their queries from trigger checking;
- A global context object providing information about a query, in particular its expected costs.

To the best of our knowledge there is no other approach published which conveys the same power, flexibility, and ease of use. Notably, this concept only affects authorization and still allows any sort of external authentication (such as SAML tokens, LDAP, etc.).

These array triggers are implemented in rasdaman, thereby validating implementability. Next, we will apply access control in several projects (such as BigDataCube [6] and Landsupport [7]) and with various rasdaman operators from industry and academia. In particular from the several rasdaman-based Earth data federations already active we expect valuable practical input.

Preliminary experiments have shown that triggers per se do not impose a particular extra load to the system. Checking for applicable triggers adds some negligible constant overhead. Checking trigger constraints depends on their complexity, but undergoes the same optimization scrutiny as all regular queries. Moreover, array tiles loaded for constraint evaluation and subsequently needed again for query processing remain in cache, so obviously constraint checking usually does not cause extra disk access.

A challenge is that checks must be performed a-priori whereas exact information about the costs incurred is only available a-posteriori. In particular for procedural interfaces like python based tools (such as, e.g., the Australian Data Cube) this is hard, if not impossible, to achieve. Fortunately, the cost-based optimizer of a datacube query system like rasdaman provides such information already, and – as first, preliminary tests show – at an acceptable quality.

In rasdaman OGC-based WMS, WCS; and WCPS Web APIs are offered where users see objects with their spatio-temporal coordinates, not the Cartesian index coordinates of the internal array. All such requests are uniformly translated into "Array SQL" queries internally, so they automatically are subject to access control. Ongoing implementation will transform the access control information to geo Web service level, in particular: allowing administrators to define limitations in geo/time coordinates, rather than pixel coordinates.

ACKNOWLEDGMENT

This work is being supported by H2020 LandSupport, H2020 EOSC-hub, and German BMWi BigDataCube.

REFERENCES

[1] Baumann, P., Rossi, A.P., Misev, D., Dumitru, A., Merticariu, V., Huu, B.P., Figuera, R.M., Kakaletris, G., Koltsida, P., Siemen, S., Wagemann, J., Clements, O., Hogan, P., "Big Earth Data at Your Fingertips", In: Mathieu, P.P., Aviset, C. (eds.): Earth Observation Open Science and Innovation, ISSI Publications 2017, pp. 91 – 119

[2] Baumann, P., Misev, D., Merticariu, V., Pham Huu, B., "Datacubes: Towards Space/Time Analysis-Ready Data", In: J. Doellner, M. Jobst, P. Schmitz (eds.): Service Oriented Mapping - Changing Paradigm in Map Production and Geoinformation Management, Springer Lecture Notes in Geoinformation and Cartography, 2018

[3] Baumann, P., Misev, D., Merticariu, V., Pham Huu, B., Bell, B., "rasdaman: Spatio-Temporal Datacubes on Steroids", Proc ACM SIGSPATIAL, Seattle, USA, November 7, 2018

[4] ISO, "Information technology — Database languages — SQL — Part 15: Multi-Dimensional Arrays", ISO 9075-15:2018

[5] ISO, "Information technology — Database languages — SQL — Part 2: Foundation", ISO 9075-2:2016

[6] N.n., "BigDataCube", http://www.bigdatacube.org/

[7] N.n., "Landsupport", https://www.landsupport.eu/

[8] N.n., "Australian Geoscience Data Cube", http://www.datacube.org.au/

[9] N.n., "SciDB Security Model Overview", https://paradigm4.atlassian.net/wiki/spaces/scidb/pages/421068858/Security+Model+Overview

[10] N.n.,"Hyrax: Support File-level access control" https://github.com/samvera/hyrax/issues/1621