Trusty URIs: Verifiable, Immutable, and Permanent Digital Artifacts for Linked Data

#eswc2014Kuhn

Tobias Kuhn and Michel Dumontier

http://www.tkuhn.ch / http://dumontierlab.com
@txkuhn / @micheldumontier

ETH Zurich / Stanford University

ESWC
27 May 2014

Motivation 1

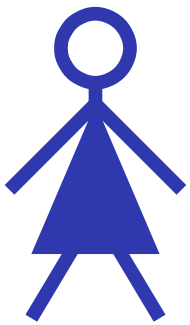
The Semantic Web: Web content becomes machine-interpretable.

Machines (i.e. algorithms) can then perform — on large amounts of linked data — tasks such as: *automated aggregation*, *complex searches*, *problem solving*, *recommendations*, *and much more ...*

But wait... even human users are often easy to trick by spam and fraudulent content found on the web. **We should be even more concerned in the case of machines!**

Motivation 2

Sue publishes a script that allows everybody to replicate her scientific analysis:



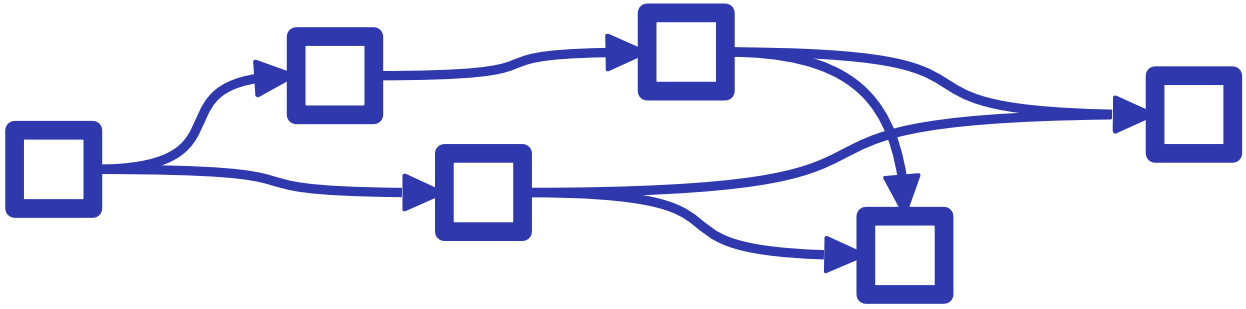
```
# Download data:
wget http://some-third-party.org/dataset/1.4
# Analyze data
...
```

But what if the third party silently changes that version of the dataset? What if the resource becomes unavailable at this location? What if the web site later gets hacked and the data manipulated?

Motivation 3

Nanopublications: Atomic pieces of scientific results together with their provenance, all represented in RDF.

- Citation networks: nanopubs can cite or refer to other nanopubs
- Nanopubs are supposed to be immutable

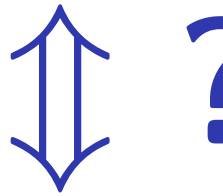


Problem:

- A scientist citing something wants to be sure that it is not silently changed afterwards
- The current web has no mechanism to enforce immutability

Problem

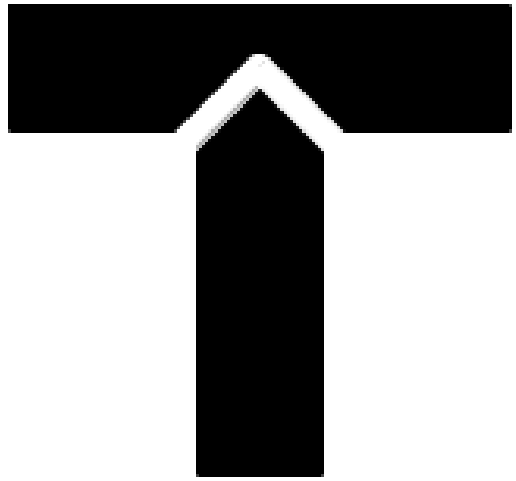
`http://some-third-party.org/dataset/1.4`



?

Given a URI for a digital artifact, there is no reliable standard procedure of checking whether a retrieved file really represents the correct and original state of that artifact.

We need URIs we can Trust!



Trusty URIs

Trusty URIs

Basic idea: Use of cryptographic hash values calculated on digital artifacts.

Requirements:

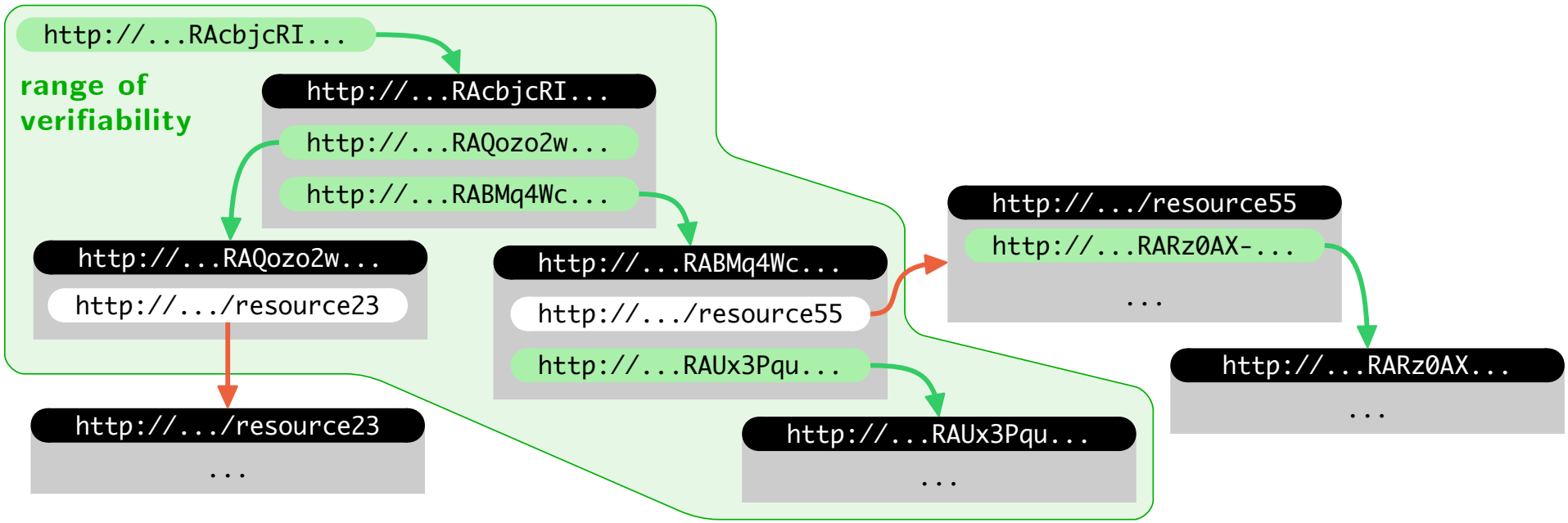
- To allow for the verification of entire reference trees, the hash should be part of the reference (i.e. the URI)
- To allow for meta-data, digital artifacts should be allowed to contain self-references (i.e. their own URI)
- Format-independent hash for different kinds of content
- The complete approach should be decentralized and open
- We want to use them right away

Example:

`http://example.org/r1.RA5AbXdpz5DcaYXCh9l3eI9ruBosiL5XDU3rxBbBaUO70`

Trusty URIs: Range of Verifiability

With the hash as a part of the URI, the "range of verifiability" extends to referenced artifacts (if they also use trusty URIs):



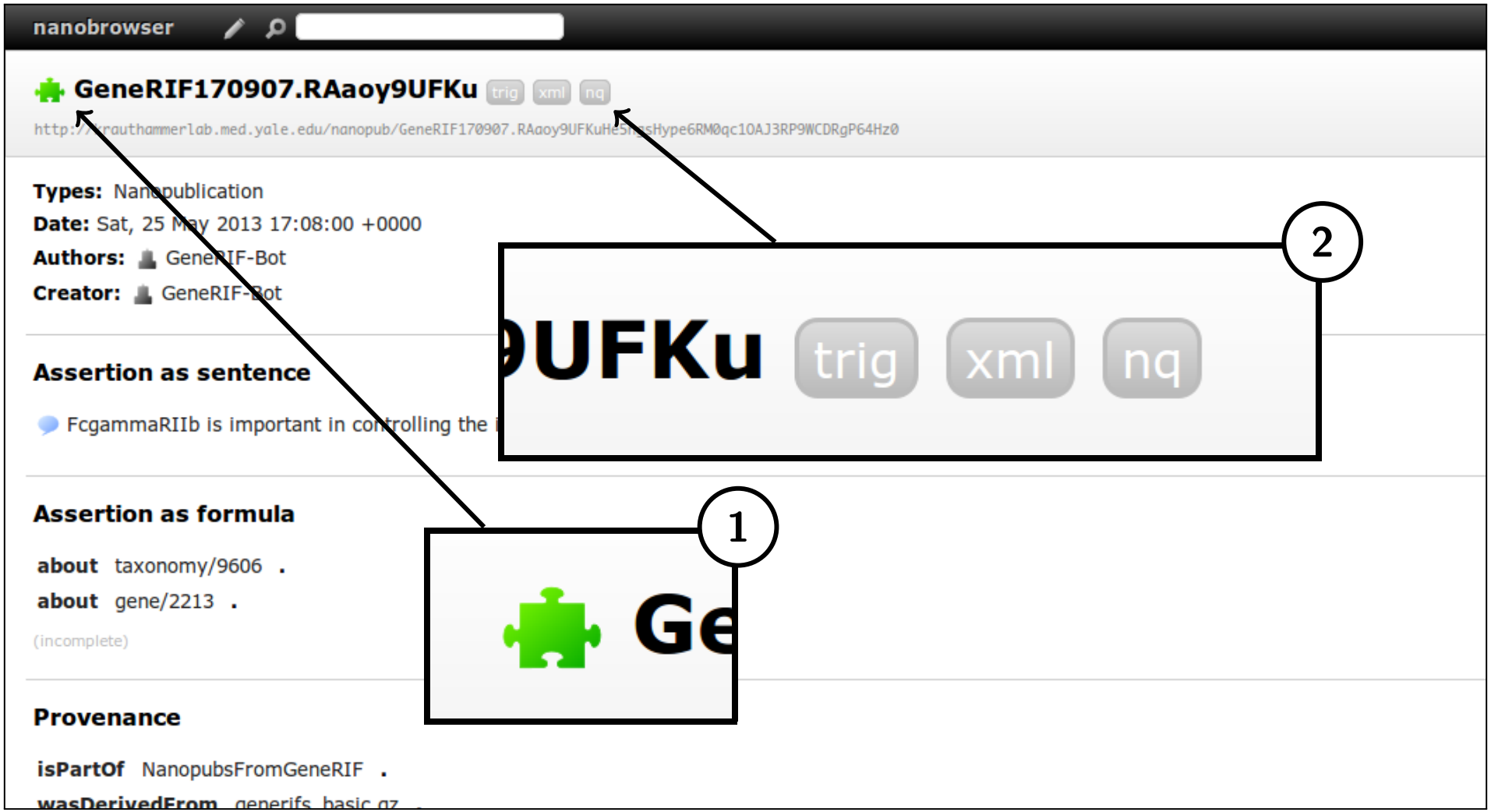
Trusty URI Modules

Currently, there are two trusty URI modules:

- **FA:** Plain files (i.e. byte sequences)
- **RA:** Sets of RDF graphs
- More to come in the future...

The first character (**F** or **R**) represents the type of the module; the second character (**A**) its version.

Example: Nanobrowser



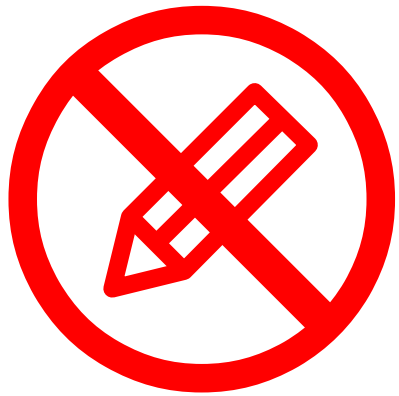
http://nanobrowser.inn.ac

Verifiable — Immutable — Permanent

Whether or not a given resource is the one a given trusty URI is supposed to represent can be **verified with perfect confidence**.

(assuming that the trusty URI for the required artifact is known, e.g. because another artifact contains it as a link)

Verifiable — **Immutable** — Permanent



Trusty URI artifacts are **immutable**, as any change in the content also changes its URI, thereby making it a new artifact.

(as soon as your trusty URI has been picked up by third parties, e.g. cached or linked from other resources, every change will be noticed)

Verifiable — Immutable — **Permanent**

Trusty URI artifacts are **permanent**, as they can be retrieved from the cache of third-party websites if otherwise no longer available.

(if there are search engines and web archives regularly crawling and caching the artifacts on the web)

Permanent Digital Artifacts

Ideally, a (trusty) artifact should be retrievable via its URI:

$\Rightarrow$ `http://my-organization.org/datasets/RA5AbX...`

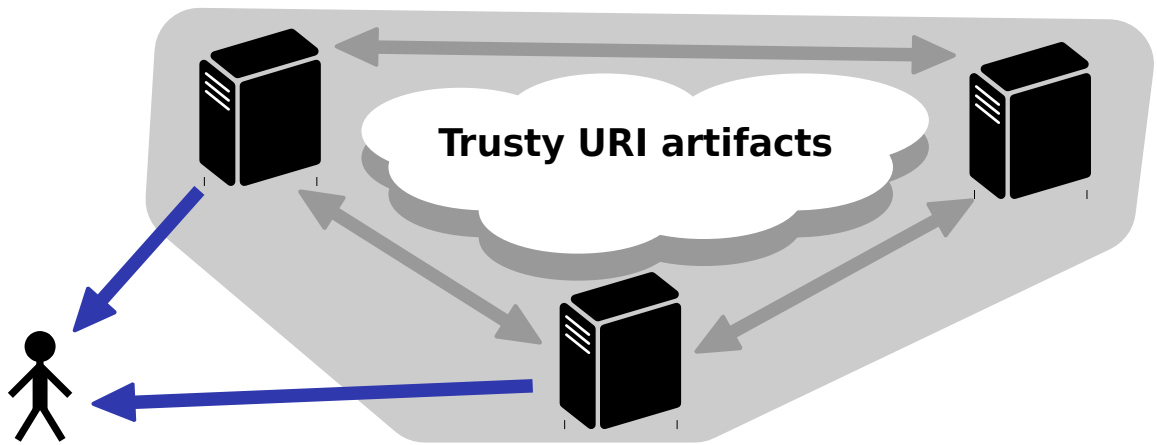
But if not, we can also retrieve it from third-party sources:

$\not\Rightarrow$ `http://my-organization.org/datasets/RA5AbX...`

$\Rightarrow$ `http://hashcache.org/object/RA5AbX...`

$\Rightarrow$ `http://artifact-archive.com/artifacts/RA5AbX...`

$\Rightarrow$ `http://nasty-server.com/no-need-to-trust-me/RA5AbX...`



Implementations

(Partial) Implementations in:

- Java (`https://github.com/trustyuri/trustyuri-java`)
- Python (`https://github.com/trustyuri/trustyuri-python`)
- Perl (`https://github.com/trustyuri/trustyuri-perl`)
- *more to come...*

Functions:

- General: CheckFile, RunBatch
- Module FA only: ProcessFile
- Module RA only: TransformRdf, TransformLargeRdf, TransformNanopub, CheckLargeRdf, CheckSortedRdf, CheckNanopubViaSparql
- *more to come...*

Evaluation 1: Nanopubs

We took ∼150,000 nanopublications from previous work, transformed them to different formats (TriG, N-Quads, and TriX), and then generated trusty URIs for them.

⇒ For any given nanopub, the same trusty URI was generated for the different formats

Then we checked these trusty URIs, also for corrupted copies of the files (one random byte changed).

⇒ All non-corrupted files are successfully validated

⇒ All corrupted files either lead to errors or the validation fails (except for <1% harmless cases in TriX format where the changed byte is not part of the RDF content)

⇒ Checking with Java in batch mode takes 0.001s per nanopub

Evaluation 2: Bio2RDF

To evaluate our approach on larger files, we transformed and checked 858 RDF files from Bio2RDF.

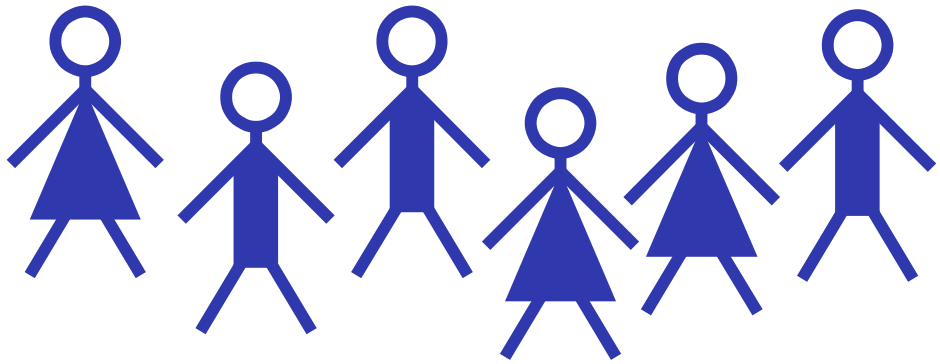
- File sizes ranging from 1.4kB to 177GB

⇒ Files smaller than 10MB require less than 3 seconds to be transformed or checked

⇒ Large files of 2GB require ∼5min to be transformed and ∼2min to be checked

⇒ Largest file of 177GB (much larger than memory) required 29h to be transformed and 3h to be checked

Make This a Community Effort



Code on GitHub: `https://github.com/trustyuri/`

Permissive Open Source License

Open Development: Let us know if you want to be involved!

Wiki (including wish list):
`https://github.com/trustyuri/trustyuri/wiki`

Conclusions and Future Work

Contribution:

- Unambiguous URI references for verifiable, immutable, and permanent digital artifacts
- Proposal of a central technical pillar of the (semantic) web
- In particular for scientific data, where provenance and verifiability are crucial

Planned usage:

- Next version of Bio2RDF
- Nanopublications for neXtProt (currently ∼20 million nanopubs)
- Nanopub server (for publishing and archiving nanopubs)

Thank you for your Attention!

Twitter: @txkuhn and #eswc2014Kuhn

Web: http://trustyuri.net

Some Additional Slides Follow...

Related Approaches

Git inspired the design of trusty URIs: Git refers to commits by hash values calculated in a recursive way.

Named Information (ni) URIs:

```
ni:///sha-256;UyaQV-Ev4rdLoHyJJWCi11OHfrYv9E1aGQAlMO2X_-Q
```

(Trusty URIs can be mapped to ni-URIs.)

What is **missing** in these approaches:

- Digital artifacts on a more abstract level than byte sequences
- Support for self-references

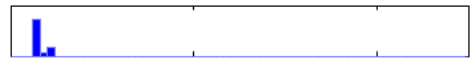
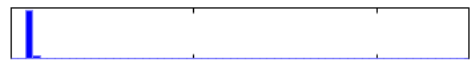
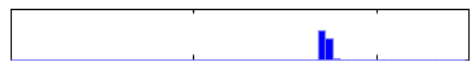
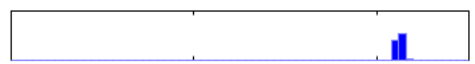
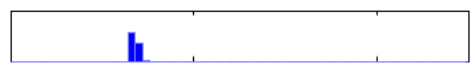
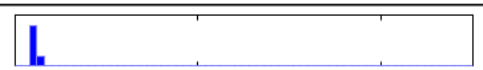
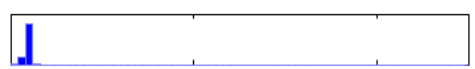
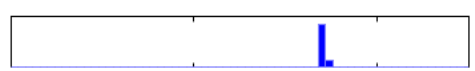
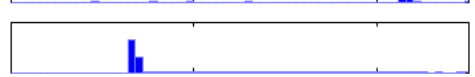
Skolemization of Blank Nodes

The hash also helps us to solve the problem of blank nodes for canonicalization of RDF content: We use the hash to skolemize blank nodes:

```
http://foo.org/r3.RACjKTA5dl23ed7JIpgPmS0E0dcU-XmWIBnGn6Iyk8B-U#_1
http://foo.org/r3.RACjKTA5dl23ed7JIpgPmS0E0dcU-XmWIBnGn6Iyk8B-U#_2
...
```

These URIs are guaranteed to have never been used before (except possibly for exactly the same content).

Performance for Nanopubs in Batch Mode

	method		time in seconds					result		
	impl.	format	mean	stdev	min	max	histogram	valid	invalid	error
valid files	Java	N-Quads	0.0019	0.0062	0.0013	1.7202		100%	0%	0%
	Java	TriG	0.0009	0.0050	0.0008	1.7412		100%	0%	0%
	Java	TriX	0.0011	0.0050	0.0009	1.5656		100%	0%	0%
	Perl	N-Quads	0.0172	0.0006	0.0171	0.0679		100%	0%	0%
	Perl	TriG	0.0214	0.0016	0.0211	0.0872		100%	0%	0%
	Python	N-Quads	0.0070	0.0011	0.0065	0.0644		100%	0%	0%
	Python	TriX	0.0070	0.0009	0.0066	0.0578		100%	0%	0%
corrupted files	Java	N-Quads	0.0012	0.0062	0.0006	1.6559		0%	99.72%	0.28%
	Java	TriG	0.0010	0.0049	0.0003	1.6335		0%	83.37%	16.63%
	Java	TriX	0.0011	0.0044	0.0005	1.3451		0.83%	84.15%	15.03%
	Perl	N-Quads	0.0171	0.0005	0.0169	0.0732		0%	100%	0%
	Perl	TriG	0.0195	0.0055	0.0007	0.0841		0%	84.49%	15.51%
	Python	N-Quads	0.0069	0.0011	0.0065	0.1716		0%	100%	0%
	Python	TriX	0.0063	0.0021	0.0006	0.1325		0.12%	84.46%	15.42%

Performance for Large Files (Bio2RDF)