Zenodo Keyword Auto-Suggest with Parallel Graph Analytics

An Anomaly Detection and Recommender System for **Zenodo** with **Oracle PGX**

Alastair Paragas

One of the noisy dudes on the bus to Zurich

github.com/alastairparagas

Team USA (Land of the Free, Home of the Brave)

Manuel Martin Marquez

Advisor

CERN OpenLab-IT-DB

What I learned this summer

Employing the use of codewords

Operation Charms, Broken Arrow

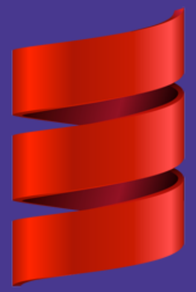
Bachelor, Bachelorette, etc

Fighting 2 consecutive bike falls

Cheese actually tastes good

Some stuff from the lectures

Hopefully

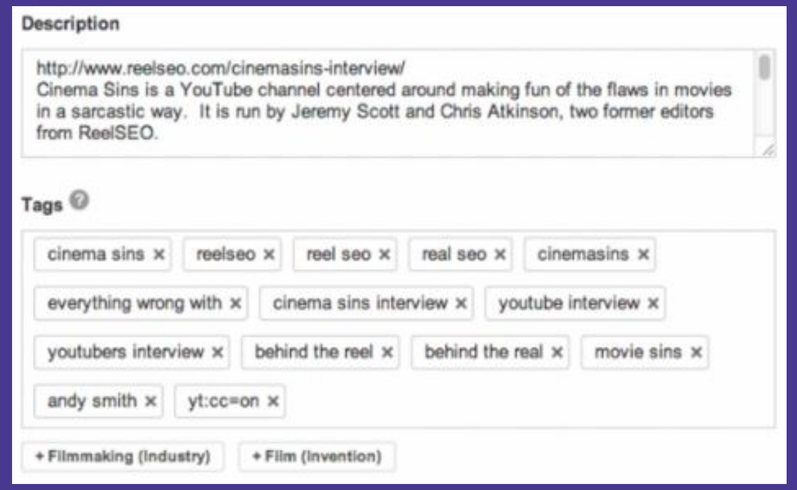


Oracle Labs
PGX

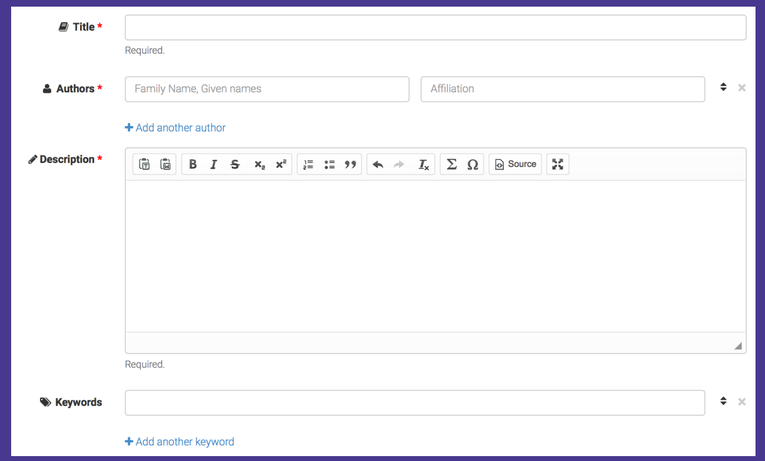
OpenLab People Suck

I just caught your attention with **fake news**. They're actually cool.

About My Project



A similar system in YouTube



Current upload form in zenodo

Asked to build a keyword suggestion system for Zenodo to make the process of "keyword tagging" a work easier.

A Recommender System

Be able to suggest contextually similar keywords to a user based on the keywords he/she has typed so far.

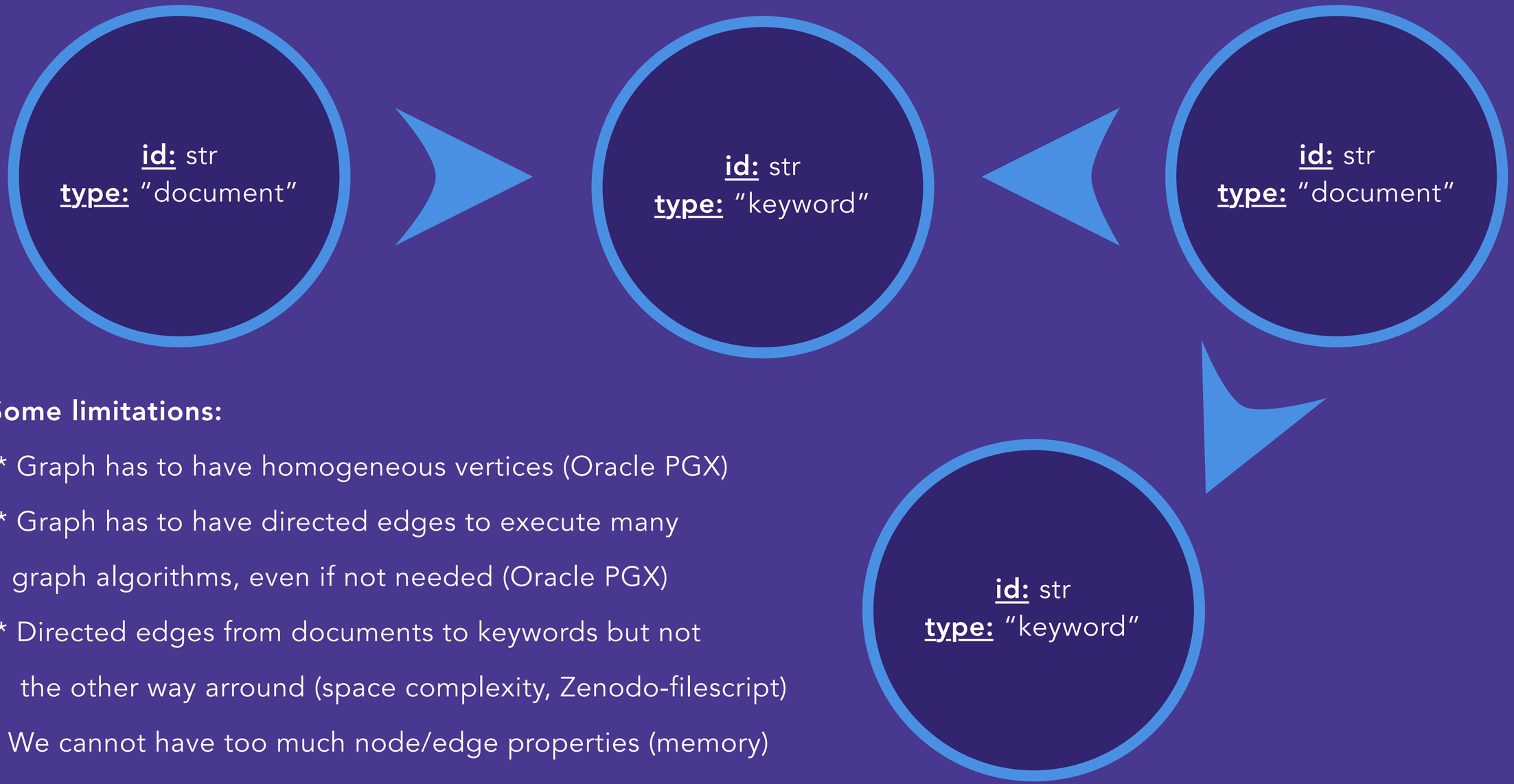
An Anomaly Detection System

Notify the user whenever he/she is potentially typing a contextually irrelevant keyword.

Test out Oracle PGX

Survey Oracle PGX's capabilities as an in-memory graph engine - allowing us to execute built-in graph algorithms or write our own graph traversal algorithms

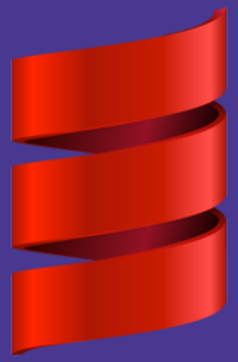
Graph Structure (dat graph though)



Some limitations:

* Graph has to have homogeneous vertices (Oracle PGX)
* Graph has to have directed edges to execute many graph algorithms, even if not needed (Oracle PGX)
* Directed edges from documents to keywords but not the other way arround (space complexity, Zenodo-filescript)
* We cannot have too much node/edge properties (memory)

Under the Hood (Don’t read me, look at the nice pictures lol)

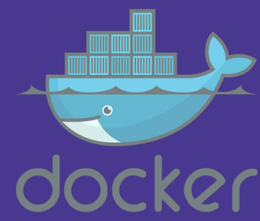


Scala - main language of choice for the actual recommender and anomaly detection system

Python - script that converts Zenodo document metadata to a flat file that is loaded into Oracle PGX

PostgreSQL - allows us to query keywords with full-text search. Oracle PGX is lacking this capability greatly.

Redis - caches our results so that we can gain speedups for computations as graph computations are costly



Docker - containerize Postgres, Redis and Scala as well as cross compile Python script for offline 2TB server

Sadly, my project did not use cool tech like **VIRTUAL REALITY**,

REAL MACHINE LEARNING or **BLOCKCHAIN W/ ETHEREUM CONTRACTS**

(But I did use **NATURAL LANGUAGE PROCESSING** ooohhh)

Under the Hood - Recommendation (if you're reading this, you're not asleep)

JSON request to /recommendation

```
{
   keyword: [string]
   normalizer: string (optional),
   ranker: string,
   vertexFinder: string (optional)
   addons: [string] (optional),
   count: int (optional)
}
```

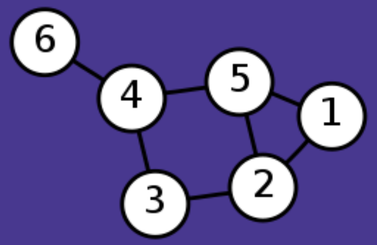
Consult cache

If the user wanted cached results, check Redis if results obtained from a previous graph computation are cached. If they are, use those cached results. If not, execute computation like normal.

Full-Text Keyword Search

We need to be able to find a keyword that is the closest possible match to an inputted keyword - fuzzy and normalized string search. The result is then used as the starting vertex for our graph algorithms

PostgreSQL



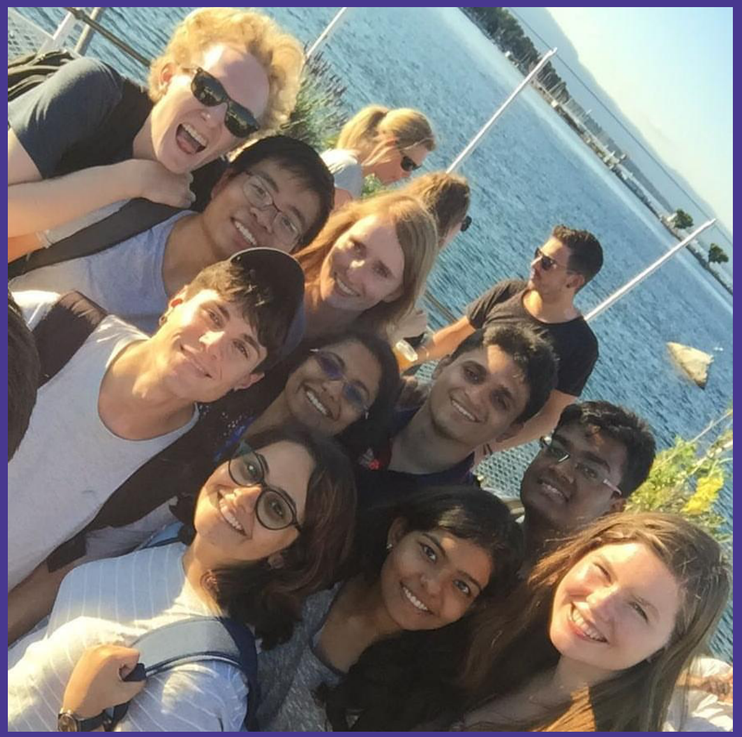
Graph Execution

Execute graph algorithms - Personalized Page Rank, Djikstra and many more to determine what keyword to suggest!

Status

Can shoot with 100% accuracy
(Source: Pierpaolo, Fete De Geneve)

Can do my 6-mile and gym exercise routine even when I'm running late the day of the Lightning Talk
(Source: this morning)



Friends with people that are just as weird as I am… or not
(Source: Swapneel's camera)

Status (To prove I'm not on Facebook all day long)

Basically, at this point, I have:

* Picked up **Oracle PGX**
* Finished **Zenodo-Filescript**
* Finished the **Recommender System**
* Have yet to finish the **Anomaly Detection System** and **Full-Text Search**

Week 1 Survey of Oracle PGX, Sample usage of Oracle PGX client-libs in Python and NodeJS, Laying out **Zenodo-Filescript**

Week 2 Finalizing **Zenodo-Filescript**, Debugging 2TB RAM Server

Week 3 Picking up Scala, laying out the architecture for **Zenodo-Addon** (graph access with concurrent web requests, execution flow)

Week 4 PPRKeywordProximityRanker and LemmaGraphNormalizer Implementation

Week 5 Debugging and optimizing PPRKeywordProximityRanker and LemmaGraphNormalizer implementations

Week 6 Refactor PPRKeywordProximityRanker to 2 different specific rankers - PPRKeywordProximityRanker and PPRKeywordProximityMeanRanker

Week 7 Refactor request execution to have decoupled add-ons that overlay over graph query execution and implement results caching add-on.

Week 8 Implement sequential keyword recommendation filtering and refactor architecture to take out how we find starting vertices (for full-text search)

Validation (You ain't just making stuff up, right?)

```
%pgx
iterator = g.queryPgql("SELECT x WHERE (x WITH type='keyword' AND id() =~
    'electron')")
    .getResults()
    .iterator()
    .collect{ it.getVertex("x") }
    .unique{a,b -> a.getId() <=> b.getId()}
```

```
==> PgxVertex[ID=Dental implants Histomorphometric analysis Interface Soft tissu
e titanium animal confocal microscopy human immunohistochemistry methodology pat
hology review scanning electron microscopy soft tissue injury surface property t
ooth implantation wound healing Animals Humans Microscopy]
==> PgxVertex[ID=electron backscatter diffraction]
==> PgxVertex[ID=inkjet-printed electronics]
==> PgxVertex[ID=electronic waste]
==> PgxVertex[ID=Scanning electron microscope]
==> PgxVertex[ID=electronic textbook]
==> PgxVertex[ID=electron micrographs.]
==> PgxVertex[ID=microelectronic sensor of pressure]
==> PgxVertex[ID=electronics]
==> PgxVertex[ID=electronic music studio]
==> PgxVertex[ID=\"X-ray photoelectron spectrocopy\"]
==> PgxVertex[ID=diagnostic electron microscopy]
==> PgxVertex[ID=X-ray free-electron lasers]
==> PgxVertex[ID=remote optoelectronic switch]
==> PgxVertex[ID=Nitrophenols Used in Optoelectronic Applications]
```

```
%pgx
iterator = g.queryPgql("SELECT y WHERE (x WITH type='keyword' AND id() =
    'electron') <-- (z WITH type='document') --> (y WITH type='keyword')")
    .getResults()
    .iterator()
    .collect{ it.getVertex("y") }
```

```
PgxVertex[ID=voltage]
==> PgxVertex[ID=early]
==> PgxVertex[ID=rhabdovirus]
==> PgxVertex[ID=microscopy]
==> PgxVertex[ID=lyssavirus]
==> PgxVertex[ID=electron]
==> PgxVertex[ID=diagnosis]
==> PgxVertex[ID=rabies]
==> PgxVertex[ID=low]
==> PgxVertex[ID=imaging]
```

```
%pgx
analyst = session.createAnalyst()
personalizedPageRank = analyst.personalizedPagerank(g, "electron", 0.00001, 0
    .85, 5000, true)
personalizedPageRank.getTopKValues(5)
```

ID	value
electron	0.15000305496985603
Animalia	0.011483491810220287
Taxonomy	0.011088694069639375
Biodiversity	0.010803899119040748
Arthropoda	0.008945226425262688

Apache Zeppelin

Practically like a Jupyter/iPython notebook for the JVM environment

Groovy

Another JVM-based language and Oracle PGX's client mode language of choice

Demo