Using databases for social scientists

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

Computational Social Science Amsterdam
30 November 2018

Today

Günther, Elisabeth; Trilling, Damian; van de Velde, Bob: But how do we store it? Data architecture in the social-scientific research process. In: Stuetzer, C.M. (Hrsg.); Welker, M. (Hrsg.); Egger, M. (Hrsg.): *Computational social science in the age of Big Data. Concepts, methodologies, tools, and applications.* Cologne: Herbert von Halem, 2018, pp. 161–187

When and why databases?

The "traditional" approach

Example: Analysis of a couple of thousands articles/speeches/reports/...

- Store as seperate `.txt` files

If metadata (beyond what can be inferred from filename and location) are important

- Store as tabular dataset (`.csv` or proprietary format)
- (possibly: store as seperate `.json` files)

The “traditional” approach

Example: Analysis of a couple of thousands articles/speeches/reports/...

- Store as seperate `.txt` files

If metadata (beyond what can be inferred from filename and location) are important

- Store as tabular dataset (`.csv` or proprietary format)
- (possibly: store as seperate `.json` files)

The “traditional” approach

Example: Analysis of a couple of thousands articles/speeches/reports/...

- Store as seperate `.txt` files

If metadata (beyond what can be inferred from filename and location) are important

- Store as tabular dataset (`.csv` or proprietary format)
- (possibly: store as seperate `.json` files)

The “traditional” approach

Example: Analysis of a couple of thousands articles/speeches/reports/...

- Store as seperate `.txt` files

If metadata (beyond what can be inferred from filename and location) are important

- Store as tabular dataset (`.csv` or proprietary format)
- (possibly: store as seperate `.json` files)

The "traditional" approach

pro

- easy to understand
- no dependencies
- works on all platforms, also in the future $\rightarrow$ good fallback/backup option

con

- inefficient
- requires loading whole dataset into memory *or* reading all files from disk to query/aggregate/etc.
- requires *you* to deal with file I/O

...

The "traditional" approach

pro

- easy to understand
- no dependencies
- works on all platforms, also in the future → good fallback/backup option

con

- inefficient
- requires loading whole dataset into memory *or* reading all files from disk to query/aggregate/etc.
- requires *you* to deal with file I/O

...

Databases: because you have to

- Your data is too big to fit in RAM (and you need to do some querying)
- Because using files would be prohibitively inefficient
- Because you need to scale horizontally

Databases: because you have to

- Your data is too big to fit in RAM (and you need to do some querying)
- Because using files would be prohibitively inefficient
- Because you need to scale horizontally

Databases: because you have to

- Your data is too big to fit in RAM (and you need to do some querying)
- Because using files would be prohibitively inefficient
- Because you need to scale horizontally

Databases: because you want to

- You do not want to take care of I/O yourself
- Because it allows you to do better searches and queries
- Because you can easily aggregate your data
- Because you can easily join/merge data
- Because you want to be able to easily modify/update records without rewriting this whole CSV table

Databases: because you want to

- You do not want to take care of I/O yourself
- Because it allows you to do better searches and queries
- Because you can easily aggregate your data
- Because you can easily join/merge data
- Because you want to be able to easily modify/update records without rewriting this whole CSV table

Databases: because you want to

- You do not want to take care of I/O yourself
- Because it allows you to do better searches and queries
- Because you can easily aggregate your data
- Because you can easily join/merge data
- Because you want to be able to easily modify/update records without rewriting this whole CSV table

Databases: because you want to

- You do not want to take care of I/O yourself
- Because it allows you to do better searches and queries
- Because you can easily aggregate your data
- Because you can easily join/merge data
- Because you want to be able to easily modify/update records without rewriting this whole CSV table

Databases: because you want to

- You do not want to take care of I/O yourself
- Because it allows you to do better searches and queries
- Because you can easily aggregate your data
- Because you can easily join/merge data
- Because you want to be able to easily modify/update records without rewriting this whole CSV table

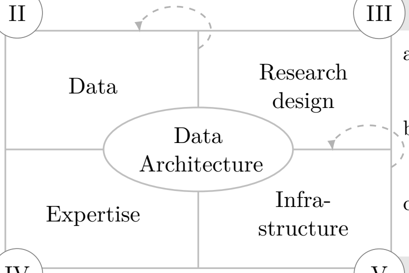
I *Do you need pre-processing / cleaning?*

II *What's your input?*

a) Technical properties (data types)
b) Source properties (static vs. dynamic)
c) Corpus properties (quantity, messiness homogeneity)

III *What's your goal?*

a) Project setup (set up an archive vs. specific research interest)
b) Project priorities (availability, consistency, performance)
c) Scientific standards (accessibility, sustainability)



IV *What are your skills?*

a) Previous experiences
b) External support (collaboration vs. contracting)
c) Future demands

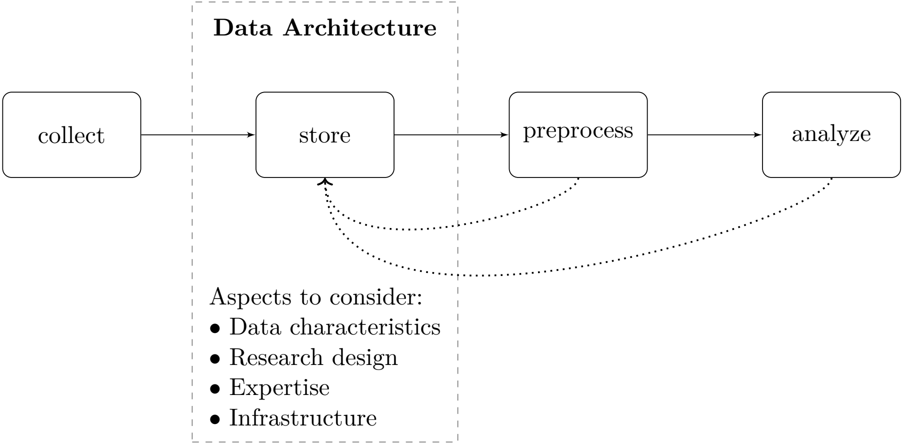
V *What else do you use?*

a) Software compatibility (technical interface)
b) Hardware scalability (vertical vs. horizontal)

Data architecture

Data architecture

- File formats
- Linkage
- Internal structure


Data Architecture
collect
store
preprocess
analyze
Aspects to consider:
• Data characteristics
• Research design
• Expertise
• Infrastructure

Database types

Database types

Relational databases

- Tabular data structure
- Relational: tables are linked by keys
- Well-defined data types per column
- Good at joining and aggregating

Relational databases

- Tabular data structure
- Relational: tables are linked by keys
- Well-defined data types per column
- Good at joining and aggregating

Relational databases

- Tabular data structure
- Relational: tables are linked by keys
- Well-defined data types per column
- Good at joining and aggregating

Relational databases

- Tabular data structure
- Relational: tables are linked by keys
- Well-defined data types per column
- Good at joining and aggregating

itemid	orderid	item	amount
5	1	Chair	200.00
6	1	Table	200.00
7	1	Lamp	123.12

customerid	name	email
5	Rosalyn Rivera	rosalyn@adatum.com
6	Jayne Sargent	jayne@contoso.com
7	Dean Luong	dean@contoso.com

orderid	customerid	date	amount
1	4	11/1/17	523.12
2	3	11/15/17	32.99
3	1	11/21/17	23.99

Relational databases and text

- You can store text
- In fact, often used as backend for for instance tweets and sometimes news articles
- BUT (1): Not optimized for searching in text
- BUT (2): Not good for messy data
- BUT (3): Hard to add extra columns or change specifications of existing ones

Relational databases and text

- You can store text
- In fact, often used as backend for for instance tweets and sometimes news articles
- BUT (1): Not optimized for searching in text
- BUT (2): Not good for messy data
- BUT (3): Hard to add extra columns or change specifications of existing ones

Relational databases and text

- You can store text
- In fact, often used as backend for for instance tweets and sometimes news articles
- BUT (1): Not optimized for searching in text
- BUT (2): Not good for messy data
- BUT (3): Hard to add extra columns or change specifications of existing ones

Relational databases and text

- You can store text
- In fact, often used as backend for for instance tweets and sometimes news articles
- BUT (1): Not optimized for searching in text
- BUT (2): Not good for messy data
- BUT (3): Hard to add extra columns or change specifications of existing ones

Relational databases and text

- You can store text
- In fact, often used as backend for for instance tweets and sometimes news articles
- BUT (1): Not optimized for searching in text
- BUT (2): Not good for messy data
- BUT (3): Hard to add extra columns or change specifications of existing ones

NoSQL databases

- optimized for messy data
- can be schema-free ⇒ we do not have to enforce a specific format at insertation time
- new entries do not necessarily have to follow the specifications of old ones
- Allow you to just throw in an arbitrary JSON object
- CAP-theorem: we trade consistency for availability and performance

NoSQL databases

- optimized for messy data
- can be schema-free ⇒ we do not have to enforce a specific format at insertation time
- new entries do not necessarily have to follow the specifications of old ones
- Allow you to just throw in an arbitrary JSON object
- CAP-theorem: we trade consistency for availability and performance

NoSQL databases

- optimized for messy data
- can be schema-free $\Rightarrow$ we do not have to enforce a specific format at insertation time
- new entries do not necessarily have to follow the specifications of old ones
- Allow you to just throw in an arbitrary JSON object
- CAP-theorem: we trade consistency for availability and performance

NoSQL databases

- optimized for messy data
- can be schema-free ⇒ we do not have to enforce a specific format at insertation time
- new entries do not necessarily have to follow the specifications of old ones
- Allow you to just throw in an arbitrary JSON object
- CAP-theorem: we trade consistency for availability and performance

NoSQL databases

- optimized for messy data
- can be schema-free $\Rightarrow$ we do not have to enforce a specific format at insertation time
- new entries do not necessarily have to follow the specifications of old ones
- Allow you to just throw in an arbitrary JSON object
- CAP-theorem: we trade consistency for availability and performance

We can store retrieved web pages (1) together with some first roughly parsed extracted data (2), and do some cleaning and enrichment (3,4) later.

1. Retrieved →

```
<div>
<span id='author'
class='big'>
Author: John
Doe </span>
<span id='viewed'>
seen 42 times
</span>
</div>
```

2. Structured →

```
{
"author":
"Author: John Doe",
"viewed":
"seen 42 times"
}
```

3. Cleaned →

```
{
"author":
"john doe",
"viewed":
42,
}
```

4. Enriched

```
{
"author":
"john doe",
"author_gender":
"M",
"viewed":
42,
}
```

NoSQL databases and text

- Often optimized for indexing text
- Internal preprocessing ("analysis") under the hood: e.g., search based on stemmed text
- Nested data: e.g., comments within articles
- Texts can have additional keys that others don't (e.g., online news have urls, offline news have page numbers; some may have)
- You may have even different language versions of the same text (and all having analyzed accordingly)

NoSQL databases and text

- Often optimized for indexing text
- Internal preprocessing ("analysis") under the hood: e.g., search based on stemmed text
- Nested data: e.g., comments within articles
- Texts can have additional keys that others don't (e.g., online news have urls, offline news have page numbers; some may have)
- You may have even different language versions of the same text (and all having analyzed accordingly)

NoSQL databases and text

- Often optimized for indexing text
- Internal preprocessing ("analysis") under the hood: e.g., search based on stemmed text
- Nested data: e.g., comments within articles
- Texts can have additional keys that others don't (e.g., online news have urls, offline news have page numbers; some may have)
- You may have even different language versions of the same text (and all having analyzed accordingly)

NoSQL databases and text

- Often optimized for indexing text
- Internal preprocessing ("analysis") under the hood: e.g., search based on stemmed text
- Nested data: e.g., comments within articles
- Texts can have additional keys that others don't (e.g., online news have urls, offline news have page numbers; some may have)
- You may have even different language versions of the same text (and all having analyzed accordingly)

NoSQL databases and text

- Often optimized for indexing text
- Internal preprocessing ("analysis") under the hood: e.g., search based on stemmed text
- Nested data: e.g., comments within articles
- Texts can have additional keys that others don't (e.g., online news have urls, offline news have page numbers; some may have)
- You may have even different language versions of the same text (and all having analyzed accordingly)

SQL

- You know the structure in advance
- You can differentiate between short strings (e.g., names) stored as VARCHAR and long strings (e.g., articles) stored as TEXT
- You expect that you don't need to query on the TEXTs, which are *not* hold in memory (and need to be fully scanned)
- Single source of truth is important, no duplication, consistency to be avoided

NoSQL

- You do not know the (full) structure in advance
- Full-text search (including preprocessed ('analyzed') version) is relevant
- You want to add new keys as you go (e.g., store preprocessing results or enrichments (sentiment scores, predictions))
- store first, clean up later

SQL

- You know the structure in advance
- You can differentiate between short strings (e.g., names) stored as VARCHAR and long strings (e.g., articles) stored as TEXT
- You expect that you don't need to query on the TEXTs, which are *not* hold in memory (and need to be fully scanned)
- Single source of truth is important, no duplication, consistency to be avoided

NoSQL

- You do not know the (full) structure in advance
- Full-text search (including preprocessed ('analyzed') version) is relevant
- You want to add new keys as you go (e.g., store preprocessing results or enrichments (sentiment scores, predictions))
- store first, clean up later

MongoDB and Elastic Search

Our use case

- Scrape and store articles ($\approx$ 20M)
- We first used Mongo and later switched to ES (because of better performance for full-text search)
- On the other hand, there are some who say MongoDB is 'safer' (less risk of data loss)

Our use case

- Scrape and store articles ($\approx$ 20M)
- We first used Mongo and later switched to ES (because of better performance for full-text search)
- On the other hand, there are some who say MongoDB is 'safer' (less risk of data loss)

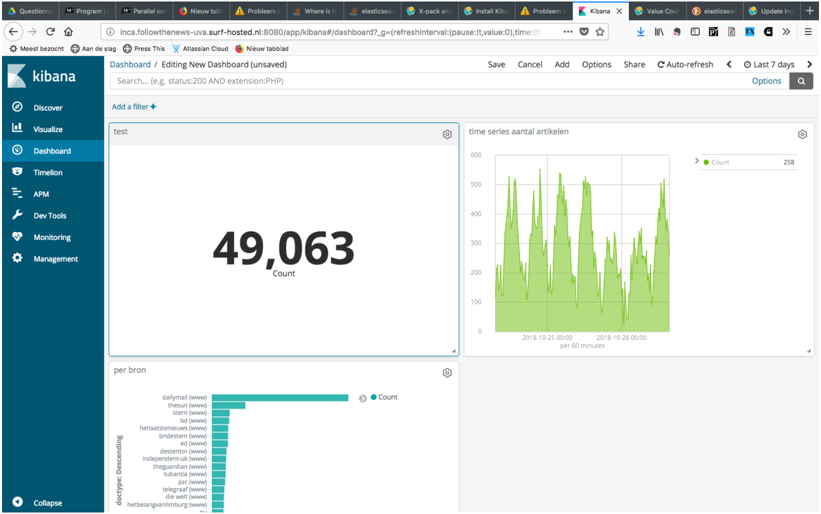
Our use case

- Scrape and store articles ($\approx$ 20M)
- We first used Mongo and later switched to ES (because of better performance for full-text search)
- On the other hand, there are some who say MongoDB is 'safer' (less risk of data loss)

Interacting with ES via http requests

```
packer-ubuntu-16:~$ curl http://localhost:9200/inca6/_count?pretty
{
  "count" : 19054530,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  }
}
```

Interacting with ES via Kibana



Interacting with ES via Python

```
In [1]: from elasticsearch import Elasticsearch

In [2]: client = Elasticsearch()

In [3]: client.count('inca6')
Out[3]:
{'_shards': {'failed': 0, 'skipped': 0, 'successful': 5, 'total': 5},
 'count': 19054530}
```

Practical example (Jupyter Notebook)

Questions?

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net