Database infrastructure for electronic structure calculations

Fawzi Mohamed
fawzi.mohamed@fhi-berlin.mpg.de

22.7.2015
Why should you be interested in databases?

- Can you find a calculation that you did one year ago? two?
- Do you think you can use part of your calculations for other purposes?
- A database can help to collect, search & organize the data
NoMaD Repository

- repository for raw data of (ab-initio) calculations
- open access data (and restricted for limited time)
- sharing of calculations
- verification of published data
- discover and repurpose calculations

http://nomad-repository.eu/
NoMaD Repository Movie

https://www.youtube.com/watch?v=L-nmRSH4NQM
NoMaD Repository
NoMaD Repository

Structure
- Dimension:
  - atom
  - 0D
  - 1D
  - 2D
  - 3D
- Crystal system:
  - cubic
  - tetragonal
  - hexagonal
  - rhombohedral
  - orthorhombic
  - monoclinic
  - triclinic
- System type:
  - adsorption
  - amino acid
  - organic
  - perovskite

Chemical elements
- Elements:
  - H
  - He
  - Li
  - Be
  - B
  - C
  - N
  - O
  - F
  - Ne
  - Na
  - Mg
  - Al
  - Si
  - P
  - S
  - Cl
  - Ar
  - K
  - Ca
  - Sc
  - Ti
  - V
  - Cr
  - Mn
  - Fe
  - Co
  - Ni
  - Cu
  - Zn
  - Ga
  - Ge
  - As
  - Se
  - Br
  - Kr
  - Rb
  - Sr
  - Y
  - Zr
  - Nb
  - Mo
  - Tc
  - Ru
  - Rh
  - Pd
  - Ag
  - Cd
  - In
  - Sn
  - Sb
  - Te
  - I
  - Xe
  - Cs
  - Ba
  - La
  - Ce
  - Pr
  - Nd
  - Pm
  - Sm
  - Eu
  - Gd
  - Tb
  - Dy
  - Ho
  - Er
  - Tm
  - Yb
  - Lu
  - Hf
  - Ta
  - W
  - Re
  - Os
  - Ir
  - Pt
  - Au
  - Tl
  - Pb
  - Bi
  - Po
  - At
  - Rn
  - Fr
  - Ra
  - Lr
  - Rf
  - Db
  - Sg
  - Bh
  - Hs
  - Mt
  - Ds
  - Rg
  - Cn
  - Uu
  - Fl
  - Uu
  - Lv
  - Uu

Methodology
- Code:
  - octopus
  - Abinit
  - CASTEP
  -CRYSTAL
  - exxtl
  - FHI-aims
  - Gaussian
  - Quantum Espresso
  - VASP
  - WEn2K
- Basis set type:
  - numeric AOs
  - gaussian
  - (L)APW+lo
- xc treatment:
  - MP
  - GW
  - CC
  - CI
  - LDA
  - GGA
  - hybrid
  - meta-GGA
  - +U
  - vdW
  - HF
  - unknown

Authorship
- Data access:
  - restricted
  - open access

Identifier
- DOI or web address search:
  - Search
<table>
<thead>
<tr>
<th>Object</th>
<th>Formula units</th>
<th>Dimension</th>
<th>Space group</th>
<th>Basis set type</th>
<th>Code</th>
<th>Upload date</th>
<th>Authors</th>
<th>Data access</th>
<th>Reference</th>
<th>Comment</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mo₃AgB₁</td>
<td>3D</td>
<td>plane waves</td>
<td>PBE</td>
<td>31/03/15</td>
<td>S. Curtarolo, F. Rose, R. Taylor, O. Levy, C. Toher, M. Nardelli, W. Setyawan, S. Wang, J. Xue, K. Yang, L. Nelson, G. Hart, S. Sarvino, N. Mingo</td>
<td>link</td>
<td>527.3KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mo₃AgB₁</td>
<td>3D</td>
<td>plane waves</td>
<td>PBE</td>
<td>31/03/15</td>
<td>S. Curtarolo, F. Rose, R. Taylor, O. Levy, C. Toher, M. Nardelli, W. Setyawan, S. Wang, J. Xue, K. Yang, L. Nelson, G. Hart, S. Sarvino, N. Mingo</td>
<td>link</td>
<td>652.7KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NoMaD Repository

- python parser scans run outputs and fills a database
- local version planned in two months

http://nomad-repository.eu/
NoMaD Center of Excellence

▶ Novel Material Discovery
▶ 4 computing centers and 8 leading scientific institution
▶ starts in November
▶ Encyclopedia
  ▶ data of materials
▶ Laboratory
  ▶ extract new information from open access data
▶ visualization of data

http://nomad-coe.eu/
NoMaD laboratory Base Layer

- Conversion layer to a code independent representation
- basis for all further analysis in the laboratory
- Data classification
  - meta data: describe all data stored within a flexible data model
- Data storage
  - json
  - HDF 5
- results of this work might be useful also to you
Store the data

- Store the data explicitly, prefer standard formats
- Disk is slow, ok to not store data that can be recreated, but in general try to err on the side of string more rather than less
- Most of the time a bit part of this work has already done for you by the program developers
- When using scripts, or means to automatize calculations like ASE do think about this
- Also useful to verify understand your data after some time or cross check your own data
Relational databases

- tables of uniform records (rows) all with the same fields (columns)
- primary key is a unique field used to access a single row (record)
- relationships (connections, references between tables)
  - one to one
  - one to many
  - many to many
- constraints
- transactions ACID (Atomicity, Consistency, Isolation, Durability)
Bank account example

accounts

<table>
<thead>
<tr>
<th>account_nr</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>12.00</td>
</tr>
<tr>
<td>22</td>
<td>3.50</td>
</tr>
<tr>
<td>33</td>
<td>43.5</td>
</tr>
</tbody>
</table>

Constraints

- account_nr is unique
- amount > 0 (no overdraw)
Transactions

1. transfer 10.00 from account 11 to account 22
   1.1 amount of 11 = (amount of 11) - 10.00
   1.2 amount of 22 = (amount of 22) + 10.00

2. transfer 10.00 from 11 to 33
   2.1 amount of 11 = (amount of 11) - 10.00
   2.2 amount of 33 = (amount of 33) + 10.00
Atomicity

- Each transaction either happens fully or not at all:
- it is not possible that the money is removed from an account and not added to the other
- or (even worse for the bank) that it is added and not removed
Consistency

- each transition goes from a valid state (satisfying the constraints) to another valid state
- only one transition can have success, not both because then the constraint that amount has to be positive would become invalid
Isolation

- guarantees that concurrent transactions are isolated from each other, in the sense that the result is consistent with a well defined sequence.
- parallel execution keeps read and writes consistent with a unique serialization
- it is not possible that both transaction 1 and 2 read the amount 12.00 from account 11 and both update it to 2.0 and thus both succeed.
Durability

- once a transaction has been committed, it will remain so
- If the transaction 1 is confirmed as committed even a crash and recovery will keep it as committed.
- often one needs a special setup for this to be really guaranteed in all cases.
### Sample DB

<table>
<thead>
<tr>
<th>runs</th>
<th>run_id</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>calculation1/out</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>evaluations</th>
<th>ev_id</th>
<th>run_id</th>
<th>energy</th>
<th>xc_f</th>
<th>str_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-123.1234</td>
<td>BLYP</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>-122.9877</td>
<td>PBE</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>eval_basis</th>
<th>ev_id</th>
<th>b_set_id</th>
<th>local_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>O</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>structures</th>
<th>str_id</th>
<th>coord</th>
<th>cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H 0...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>basis_sets</th>
<th>b_set_id</th>
<th>a_kind</th>
<th>name</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H</td>
<td>light</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>O</td>
<td>light</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
ER model

- describe the structure of the database
  - tables, columns and relationships
- prefer lowercase names with “_”
- large or difficult to recreate DB:
  - migrations to handle structural changes
Sample ER model

- **runs**
  - run_id: integer
  - path: varchar

- **eval_basis**
  - ev_id: integer
  - b_set_id: integer
  - local_label: varchar

- **basis_sets**
  - b_set_id: integer
  - a_kind: varchar
  - name: varchar
  - definition: varchar

- **evaluations**
  - ev_id: integer
  - run_id: integer
  - energy: float
  - xc_f: varchar
  - str_id: integer

- **structures**
  - str_id: integer
  - coord: varchar
  - cell: varchar
Sample ER model 2

- **runs**
  - run_id: integer
  - path: varchar

- **eval_basis**
  - ev_id: integer
  - b_set_id: integer
  - local_label: varchar

- **basis_sets**
  - b_set_id: integer
  - a_kind: varchar
  - name: varchar
  - definition: varchar

- **evaluations**
  - ev_id: integer
  - run_id: integer
  - energy: float
  - m_id: integer
  - str_id: integer

- **structures**
  - str_id: integer
  - coord: varchar
  - cell: varchar

- **methods**
  - m_id: integer
  - xc_f: varchar
  - sic: varchar
SQL (standard query language) can be used to create, change and query both the structure and the data of a relational database.

- It basically defines relational databases.
- We look only to the query part.
- Select returns a table, first you define the column (which you can also rename), * means all columns.
SQL Example

```sql
select * from evaluations limit 100 offset 0;

select
evaluations.xc_f,
evaluations.energy as e,
struct.cell as cell
from evaluations
join structures struct on
    struct.str_id == evaluations.str_id
where
    evaluations.energy < -122.87
order by evaluations.energy
limit 10;
```
I want to use it now

- personal db
  - no server needed
  - single file
  - sqlite

- large db
  - server or cluster
  - concurrent usage
  - postgres
- no server needed
- fast, small personal db
- can replace writing to a file
- gui sqlitebrowser
- can be used from many languages
Language connectivity

- many possibilities
- exposing the relational model and sql
  - python: sqlachemy
  - java/scala: jdbc (basic), jooq
No-Sql

- departure from the relational model
  - why? to be faster, scale better, handle non uniform data (just structured data), handle graphs

- many different ways to depart from it, so not well defined
  - schemaless / non uniform data
  - automatic sharding
  - Non ACID, limited transactions, eventual consistency, ...
  - graph databases

- in memory databases
json data model 1/2

- json JavaScript Object Notation, is a simple human readable serialization format consisting only of
  - objects (dictionary, key value pairs, associative arrays),
  - arrays (heterogeneous lists),
  - numbers and strings.
- maps rather naturally to data structures of programming languages.
- xml can be used for similar purposes, but is normally much more structured and checked (schema), which is seldom the case with json
json can be handled well in the browser
  ▶ modern NoSql implementations were developed for web applications -
  ▶ JavaScript can be sand-boxed easily, sometime used in queries

▶ document based databases (json or xml)
json example

{
    "calculation_id": 1,
    "method": {
        "xc_f": "blyp"
    },
    "energy": -123.5
}

"system": {
    "coord": [ "O", 0.0, 0.0, 0.0,
                "H", 0.942, 0 , 0,
                "H", -0.235, 0.9122, 0 ],
    "cell": [[10, 0, 0],
              [0, 10, 0],
              [0, 0, 10]]
    "periodicity": [true, true, true]
}
}
relational databases are adding special handling for XML and JSON
- ACID being added to keystores and NoSql databases
- adding SQL, or SQL-like query languages to NoSQL databases
- NoSQL re-branded Not only SQL, introducing NewSQL
Map & reduce

- way to express some algorithms that makes them easy to parallelize, became popular after google article
- can work well on distributed data
- Hadoop widely used opensource framework
  - HDFS: distributed file system
Map

- apply a function on all the data
- can applied in parallel on the distributed data (bring computation to the data)
- $f(x) -> [(\text{key1, value1}),\ldots]$
Group (Shuffle)

- collect all data with the same key together
Reduce

- reduce operation processes the extracted data
  - \( r(\text{key, values}) \rightarrow [\text{result1, \ldots}] \)
- reduce operation can be quite generic, for example collecting to DB
Data-flow

- after the success of map reduce trying to generalize it
- support the map, group and reduce building blocks, but try to offer more complex pipelines to transform the data
  - if the reduce operation is something that can be applied like a tree (a sum for example)
  - then data does not need to be collected on a single node
- a good framework should take care of distributing the operations in the optimal way
- iterative methods still a bit challenging
Data-flow & generic algorithms frameworks

▶ try to support not just data-flow or stream processing but also iterative methods, graph processing and some machine learning algorithms

▶ Flink, started here in Berlin, and has a strong streaming and dataflow core, and DB like optimization

▶ Spark is a bit older, and initially did focus improving Hadoop work-flows and Extract Transform Load (ETL) work-flows
Data formats for scientific data

- For map reduce and similar approaches the way the data is stored in files and/or serialized (for stream processing) can have large impact on speed.
- None of the DB or DB inspired formats are really optimized for large numerical data.
- Geo-sciences and meteorological sciences built.
- HDF 5 to efficiently store multidimensional arrays.
  - Can store data and metadata (self describing).
  - Portable (can be used on almost any platform).
  - Available for C, C++, fortran java, python (h5py), . . .
  - Well suited for large multidimensional arrays.
Summary

- Computer simulations are producing larger and larger amounts of data.
- Organizing them and extracting insight from them is bound to become an important task.
- The NoMaD project wants to develop the technology to perform this on a large scale.
- Understanding databases:
  - Relational database model
  - Map and reduce idea and modern approaches
- Store your data:
  - Use standard formats if possible
  - HDF 5 is a good format for scientific data
Acknowledgements

- Matthias Scheffler
- Luca Ghiringhelli
- The NoMaD Repository team
- Organizers
- You for your kind attention