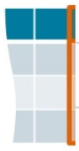
Assignment

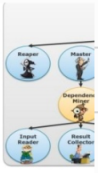


- Complete the akka-ddm project such that it discovers all unary INDs in the input datasets (within and between all tables!).
- The input dataset may consist of multiple tables.
- The solution should run distributed and exploit all available resources.
- Optional bonus task: If you seek for a challenge, also implement the "hardMode" and also discover all n-ary INDs.

Dataset

- Use the TPCH dataset provided in the git repository and on our Ilias website.
- TPCH is a generated dataset often used to benchmark query performance.

Challenges of the homework



1. Understand the available code and protocols.
2. Find an effective strategy and protocol for
   - data representation (e.g. column vs. row)
   - data distribution (e.g. full replication vs. clever partitioning).
3. Find a clever candidate generation technique (e.g. exploit pruning?).
4. Find an efficient strategy for candidate validation (e.g. valuesA.containsAll(valuesB) might not be the fastest approach).

Tasks

1. Form teams of two students.
2. Create a public GitHub repository.
3. Copy or fork the ddm-akka project from the exercise repository https://github.com/UMR-Big-Data-Analytics/ddm-akka into your repository.
4. Build, understand and test the ddm-akka project.
5. Send your first and last names, email addresses, a group name and the link of your repository via email to: vielhau4@mathematik.uni-marburg.de

Header: *DDM team <your-team-name>*

Body:
<your-team-name>
<name1> <lastname1>, <email1>
<name2> <lastname2>, <email2>
<github-URL>

Tasks (cont.)

6. Solve the homework by implementing the distributed IND discovery.
7. Test your code:
   - On one machine: by starting multiple processes locally.
   - On two machines: by connecting your two computers.
8. Document your code:
   - Create one or two PowerPoint slides to present your solution.
   - Commit the slides into your repository.

- For your team-email:

22.11.2022 23:59:00

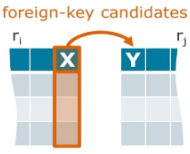
- For your final homework:

02.01.2022 23:59:00

In formationen

Definition: Given two relational instances r_i and r_j for the schemata R_i, and R_j, respectively. The inclusion dependency (IND) $R_i[X] \subseteq R_j[Y]$ (short $X \subseteq Y$) with $X \subseteq R_i$, $Y \subseteq R_j$ and $|X| = |Y|$ is valid, iff $\forall\, t_i[X] \in r_i\ ,\ \exists t_j[Y] \in r_j : t_i[X] = t_j[Y]$.

"All values in X are also contained in Y"



A unary inclusion dependency $X \subseteq Y$ is an IND with $|X| = |Y| = 1$.
An n-ary inclusion dependency $X \subseteq Y$ is an IND with $|X| = |Y| > 1$.

Axioms

- Reflexivity: If $i = j$ and $X = Y$ then $R_i[X] \subseteq R_j[Y]$.
- Permutation: If $R_i[A_1, \ldots, A_n] \subseteq R_j[B_1, \ldots, B_n]$ then $R_i[A_{\sigma 1}, \ldots, A_{\sigma n}] \subseteq R_j[B_{\sigma 1}, \ldots, B_{\sigma n}]$ for each sequence $\sigma 1, \ldots, \sigma n$ of distinct integers $\{1, \ldots, m\}$.
- Transitivity: If $R_i[X] \subseteq R_j[Y]$ and $R_j[Y] \subseteq R_k[Z]$ then $R_i[X] \subseteq R_k[Z]$.

Standardization

- Permutation axiom: Applying the same permutation σ to the both attribute lists of an IND always results in the technically same IND:
  e.g. {A,B,C} ⊆ {D,E,F} = {B,C,A} ⊆ {E,F,D} = {A,B,C} ⊆ {D,E,F} = …
- We define that the dependent attributes shall be sorted by the index in their schema, which avoids duplicate outputs in the result sets.

Inclusion Dependency Discovery

Name	Type	Equatorial diameter	Mass	Orbital radius	Orbital period	Rotation period	Confirmed moons	Rings	Atmosphere
Mercury	Terrestrial	0.382	0.06	0.47	0.24	58.64	0	no	minimal
Venus	Terrestrial	0.949	0.82	0.72	0.62	−243.02	0	no	CO_2, N_2
Earth	Terrestrial	1.000	1.00	1.00	1.00	1.00	1	no	N_2, O_2, Ar
Mars	Terrestrial	0.532	0.11	1.52	1.88	1.03	2	no	CO_2, N_2, Ar
Jupiter	Giant	11.209	317.8	5.20	11.86	0.41	67	yes	H_2, He
Saturn	Giant	9.449	95.2	9.54	29.46	0.43	62	yes	H_2, He
Uranus	Giant	4.007	14.6	19.22	84.01	−0.72	27	yes	H_2, He
Neptune	Giant	3.883	17.2	30.06	164.8	0.67	14	yes	H_2, He

Complexity: $O(n^2-n)$ for n attributes

Example:
10 attr ~ 90 checks
1,000 attr ~ 999,000 checks

- Name ⊆ Type ?
- Name ⊆ Equatorial_diameter ?
- Name ⊆ Mass ?
- Name ⊆ Orbital_radius ?
- Name ⊆ Orbital_period ?
- Name ⊆ Rotation_period ?
- Name ⊆ Confirmed_moons ?
- Name ⊆ Rings ?
- Name ⊆ Atmosphere ?
- Type ⊆ Name ?
- Type ⊆ Equatorial_diameter ?
- Type ⊆ Mass ?
- Type ⊆ Orbital_radius ?
- Type ⊆ Orbital_period ?
- Type ⊆ Rotation_period ?
- Type ⊆ Confirmed_moons ?
- Type ⊆ Rings ?
- Type ⊆ Atmosphere ?
- Mass ⊆ Name ?
- Mass ⊆ Type ?
- Mass ⊆ Equatorial_diameter
- …

ThorstenPapenbr
Slide **186**

unary inclusion dependency

Book

Title	Author	Price	Pages	Published
Database Systems	Ullman	214	1203	2007
Algorithms in Java	Sedgewick	130	768	2002
3D Computer Graphics	Watt	20	570	1999

Name ⊆ Title

Lending

ID	Name	Location	Student	Course
42	Database Systems	A-1.2	Miller	DBS 1
88	Database Systems	B-2.2	Miller	PT 1
73	Database Systems	A-1.2	Smith	DPDC
69	Algorithms in Java	C-E.1	Miller	PT 1
13	Algorithms in Java	C-E.1	Smith	DPDC

n-ary inclusion dependency

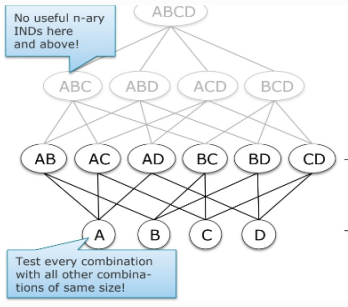
Student

Name	Lecture	Credit	Semester	Verified
Miller	DBS 1	20	2	false
Miller	PT 1	15	2	false
Smith	DPDC	10	6	true

Student, Course ⊆ Name, Lecture

Lending

ID	Name	Location	Student	Course
42	Database Systems	A-1.2	Miller	DBS 1
88	Database Systems	B-2.2	Miller	PT 1
73	Database Systems	A-1.2	Smith	DPDC
69	Algorithms in Java	C-E.1	Miller	PT 1
13	Algorithms in Java	C-E.1	Smith	DPDC



AB ⊆ CD, AB ⊆ DC
AC ⊆ BD, AC ⊆ DB
AD ⊆ BC, AD ⊆ CB
BC ⊆ AD, BC ⊆ DA
BD ⊆ AC, BD ⊆ CA
CD ⊆ AB, CD ⊆ BA

A ⊆ B, A ⊆ C, A ⊆ D
B ⊆ A, B ⊆ C, B ⊆ D
C ⊆ A, C ⊆ B, C ⊆ D
D ⊆ A, D ⊆ B, D ⊆ C

Pruning

- Upwards:
  Generate {A,B} ⊆ {C,D} only if {A} ⊆ {C} and {B} ⊆ {D} are both true!
- Downwards:
  If {A,B} ⊆ {C,D}, then {A} ⊆ {C} and {B} ⊆ {D} must also be true.
- Same level:
  If {A} ⊆ {B} and {B} ⊆ {C}, then {A} ⊆ {C} must also be true.

unary inclusion dependency = Fremdschlüssel mit 1 Wert

n-ary inclusion dependency = Fremdschlüssel mit n Werten

Gute Regeln

Coding

github /UMR-Big_Data-Analytics/ddumrakka

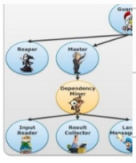
Ändere Minimum

Dependency Miner und Worker

What **to change** in the code?

- The main class should not send a shutdown on key-press, i.e. commend these lines (they are used for demo only):

```
// waitForInput(">>> Press ENTER to exit <<<");
// guardian.tell(new Guardian.ShutdownMessage());
```



- Use the DependencyMiner and DependencyWorker to implement the IND discovery. Remove all code that generated random results or pretends workload.
  - Recommendation: Master/Worker pattern ;-)
- Somehow, you need to (intelligently) transfer data to the DependencyWorkers so that they can participate in the discovery.
  - Recommendation: LargeMessageProxys

What **not to change** in the code?

- Do not change the time measurement:
  It should start with the StartMessage and end when the DependencyMiner ends the discovery with the FinalizeMessage.
- Do not change
  - the application name or packaging.
  - the application command-line interface (i.e. the jcommander parts).
- Do not use the disk!
- Let the ResultCollector manage your outputs.

What **probably/maybe to change** in the code?

- Add more actors
  (e.g. further worker actors on remote hosts or specialized child actors below the DependencyMiner for task parallelism).
- Improve the shutdown protocol for a more sophisticated downing (e.g. forward the ShutdownMessage to children of the Master and Worker actors).
- Implement the "hardMode" and discover n-ary INDs when the flag is set.
- Optimize the application.conf to your needs.

Hints

Stick to established patterns!

- Consider the patterns we learned: master/worker, proxy, router, reaper, singleton, reactive waiting, receptionist and large message proxy.

Not everything is an actor!

- Use actors where concurrency is expected; basic OO otherwise.

Use custom dispatchers!

- Isolate long running actors on dispatchers with limited amounts of threads; otherwise they starve other actors for threads.

Expect change/elasticity!

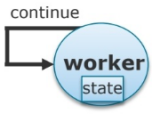
- Prepare for late joining and exiting actor systems.

Avoid mutable states in messages!

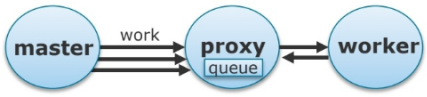
- Check that the payload of all messages is immutable (or treated as if it was) (e.g. sending mutable Java Maps lets the target actor see mutable state!).

Stay reactive!

- Actors that work on long running tasks are not reactive to e.g.
  - shutdown messages
  - pruning information
  - further work packages or requests



- Pause work frequently
  - Send a "ContinueMessage" to yourself to check your own mailbox.



- Organize help
  - Setup proxy actors to manage/queue your inbox while working long.

Pruned work beats parallelized work!

- Do not waste energy by parallelizing subtasks blindly.
- Consider your pruning potential and react on partial results to win the game!

Consider batching of messages!

- Group tasks into reasonable sized work packages.
  - Not too small (due to high messaging overhead)
  - Not too large (due to non-reactive workers)

Consider breaking of messages!

- Split large messages into reasonably sized sub-messages.
- Large refers to the message size and message computation costs.
  - Too large messages cannot be transferred (see LargeMessageProxy)
  - Too computation intensive messages make actor unresponsive (see task- and data-parallelism)

Measure, log, record and test!

- Distributed systems are difficult to **debug** (no stop-the-world breakpoints)
  - Actor hierarchies for failure debugging
  - Unit tests for actors and code modules (see Akka's `TestKit`)
- Distributed systems are hard to **profile** (limited use of JVisualVM)
  - Metrics collection for runtime profiling (see `ClusterMetricsExtension`)
  - Logging of important messages and events (see `LoggingAdapter`)

Avoid premature optimization!

- Start simple and test correctness first.
- Then, carefully identify bottlenecks and where necessary:
  - Add concurrency e.g. via routers or further child actors.

Umsetzungsidee

Analyse Manager

receive ([<TN1>,<SN1>],[<TN2>,<SN2>]) /Result/Manager

receive (Yes or No)

if No
send No
if Yes
checkIfDone

send Yes Or No → Resultmanager.add (Pair, Yes or No)
Kill Myself because I am done

checkIfDone count Done, Yes -> send Yes
checkTypes besides int, besides boolean, etc.
getSize if possible distinct
main
if checkTypes
for i in range(0, len(large)/chunkSize)
→ new ChunkAnalyse
analyse(Pair, chunkSize, i, size)
+ call MeHere
else
send No

Chunkanalyser

receive
send Yes Or No → Parent(AnalyseManager)
receive(yes or no)
Kill Myself
main
analyse

ResultManager

receive (Pair, Boolean)
add to Result

send IamDone -> MainManager.itsDone (Result)

MainManager

receiveDone (Result)

Now What?

main

Lies Tabellennamen und Spaltennamen

generiere Stack mit

[(<Tabellenname1>, <Spaltenname1>),

(<Tabellenname2>, <Spaltenname2>)]

(filter vielleicht bereits nach Spalten

if no match don't stack)

k ← new ResultManager

for each in Stack

new AnalyseManager.analyse (Pair, k)