APACHE SLING & FRIENDS TECH MEETUP
BERLIN, 25-27 SEPTEMBER 2017

Optimizing OAK repository search

O. Jentsch, Condat AG

Welcome tho the next session.
This session is: Optimizing OAK repository search.

adaptTo()

Introduction

2

Let me start with a short introduction.

Presentation Goals

- Reporting experiences made upgrading CRX2 to OAK
- Best practices for optimizing OAK repository search performance

3

My goals for this session are

- Report our experiences that we made upgrading from CRX2 to OAK at our customer systems
- Show our best practices for optimizing OAK repository search performance

About me

- Olaf Jentsch
- Senior Software Engineer at Condat AG
- working with Day CQ/AEM since 2006

4

- My name is Olaf Jentsch
- I have worked as a software enginerr at Condat AG
- I have been working with day cq, later AEM since 2006

adaptTo()

Agenda

- Customer‘s Situation
- Migrating repository search to OAK
- Measuring search improvements
- Tools

That is the agenda

- I want to start with our customers situation.
- Then I will have a long talk about migrating repository search to OAK
- After that I want to give some notes how we did the measuring of search improvements.
- At the end I want to give some hints for very useful tools that support excellent the development of search functionality.

adaptTo()

Customer‘s situation

6

Now coming to our customers situation.

adaptTo()

Migration project – initial situation

- CQ 5 / AEM 6.0 -> AEM 6.2
  - CRX2 -> OAK
- Large repository (> 1 mio documents ~ 500GB)
- Mix of (customized) XPath, SQL2 queries
- Search features should be the same
- Improving Performance

7

- They had different systems, Some of them CQ 5.6.1 and some of them AEM 6.0
- They started a migration project with AEM 6.2 as the migration target.
- So the key challenge of the migration was the change of the underlying repository from CRX2 -> OAK
- They have large repositories with more than 1 mio. documents with at least 500 GB of data
- They have their websites, their web applications containing certain search functionalities consisting of
  both xpath and sql2 queries

The declared requirements were:

- The seach features should be the same after migration
- While search performance should be improved

adaptTo()

Migrating repository search to OAK

8

Now coming to migrating repository search to OAK.

adaptTo()

Steps to migrate repository search to OAK

1. Achieve your queries to run
2. Ensure your queries to use indexes
3. Optimize index definitions
4. Optimize queries

9

We have it done in four steps.
First step: Achieve your queries to run
- In other words:
  - When you have successfully made the content migration
    with at least representative content or good test content
    try out the application specific queries, look at the system response
  - It might be that a query runs into an exception because of syntactical changes
    that have been made in the OAK implmentation
    not in the query language as a whole but in some details
  - I will show an example in the next slide

Second step: Ensure your queries to use indexes
- OAK doesn't provide an index by default for all kinds of queries
- One will notice when a query doesn't use an index, because it performs slow
- Even if it not performs too slow, one will recognize it's not using an index by a warning message
  which tell: consider creating an index
- Basically there are three options to select from what to do to reach that a query uses an index
  i will show it in coming slides

Go to step three – Optimize index definition –

- when a query still not performs fast enough,
- When you need advanced search features like fulltext search, facets, excerpts etc.
- When a query does not deliver the expected results

Step four: Optimize queries

- Often this step is done together with optimizing index definitions
- In any case you have to optimize queries when they still not perform fast enough
- I will show later some examples

adaptTo()

Query Syntax

- Adapt syntactical changes

[cond1][cond2] -> [cond1 and cond2]

```
/jcr:root/content/site//element(*)
[sling:resourceType= 'project/components/schreiben’ ][
jcr:like(@number,’prefix%’)]

/jcr:root/content/site//element(*)
[sling:resourceType= 'project/components/schreiben' and
jcr:like(@number,’prefix%’)]
```

10

At first, when you get an QueryParseException you have to adapt syntactical changes in order to achieve a query to run in OAK.
Look at this example.
It was very common in CRX2 create a complete query expression by dynamically writing filter conditions as they come from input parameters one behind the other enclosed in square brackets as a sequence.
Now one has to write explicitily the operator „and“ between the conditions and enclose all around with square brackets.
It‘s easy to make this change. First step is done very quickly.

Ensure your queries to use indexes

- Prefer out-of-the-box indexes
  - by slight changes to the query
  - by slight changes to existing out-of-the-box index
- Create your own custom indexes
  - If you search for a custom property
  - If you need aggregation, fulltext, multiple prop.

11

- Next step: ensure your queries to use indexes
- As I said before the system tells you by a warning message when a query executes without an index
- Also you can use one of the tools, the very nice „Explain Query Tool" to find out is there used an index and which one.
- Is no index used then there are basically three options to choose from what to do.
- Regarding to the more simple queries my advice is to prefer to use out-of-the-box indexes.
- One option is make only a slight change to a query so that it uses an existent index.
- Another option is slightly change an existing out-of-the-box index so that a query uses this index and gets results.
- Option three is create your own custom index.
- In the following cases it's the best option
  - When the application constantly searches for a certain custom property, something like an identifier, then create a custom property index
  - When the application needs advanced seach features like search for multiple properties, fulltext search, aggregation of properties etc

Slight changes to the query: example

adaptTo()

General nodetype change to ...

```
// works in CRX2
select [jcr:path], [jcr:score], * from [nt:base]
   as a where [rep:principalName] is not null
      and isdescendantnode(a, '/home') ...
```

... a more specific nodetype

```
// works in OAK, uses nodetype index
select [jcr:path], [jcr:score], * from [rep:User]
   as a where ...
```

12

I want to show an example for the first option.
The query above worked fine in CRX2, but is slow in OAK because it‘s a traversal query.
It was looking for a general nodetype nt:base.
Now I‘m only change the nodetype to a more specific one and already I have achieved, that this query uses the nodetype index, what is an out-of-the-box index of OAK.

adaptTo()

Slight changes to OOTB index: example

- Fulltext query combined with property condition does not work on OOTB index

```
/jcr:root//element(*, cq:Page)
[jcr:contains(jcr:content/@jcr:title, 'Rezension') and
jcr:content/@hideInNav]
```

13

- Now look at this example for the second option.
- There is a fulltext query combined with a property condition.
- The query looks for nodetype cq:Page
- The special problem is, that the query even uses the out-of-the-box index called cqPageLucene, but finds nothing,
  because the property index cqPageLucene cannot deliver results for a fulltext search.

Slight changes to OOTB index

adaptTo()

- Index definition: /oak:index/cqPageLucene

```
...
"jcrTitle": {
        "jcr:primaryType": "nt:unstructured",
        "nodeScopeIndex": true,
        "useInSuggest": true,
        "propertyIndex": true,
        "analyzed": true,
        "useInSpellcheck": true,
        "name": "jcr:content/jcr:title",
        "type": "String"
}, ...
```

14

But one can slightly change the index definition of that out-of-the-box index,
Add the fulltext property: analyzed = true
And then the query works fine without further effort.

Custom index definition

- Define index not at root level but at deeper nodes in order to
  - Restrict scope (/content/site)
  - Reduce amount of data
  - Avoid conflicts with system queries and out-of-the-box indexes

Now I want to talk about custom index definitions.
I‘m talking about definition of lucene index, which provides the more advanced search features.
My first recommendation is, define an index not at root level but at deeper nodes in the repository structure, exactly where the content of your website is located.

- So you can restrict the scope of the index and it will not be selected for queries searching for content in a different website.
- Of course your index will be the first choice for all your queries that contain path restrictions beginning with the path of your website.
- Also it reduces the amount of data needed for the index itself
- And you avoid conflicts with out-of-the-box indexes at index selection for system queries, especially when your index definition contains one of the general nodetypes like nt:base, nt:unstructured, cq:Page

adaptTo()

Custom index definition

- Include all properties used in query expression

```
/jcr:root//element(*, cq:Page)
[ not(jcr:like(jcr:content/@propA, '%that%')) and
jcr:like(jcr:content/@propB, '%this%') ]
order by jcr:content/@propC descending
```

16

My next recommendation is to include all properties used in the query expression. Look at this XPATH query, that contains three properties in different conditions.

Custom index definition

adaptTo()

```
"propA": {
        "ordered": false,
        "propertyIndex": true,
        "name": "jcr:content/propA", ...
},
"propC": {
        "ordered": true,
        "propertyIndex": true,
        "name": "jcr:content/propC", ...
},
"propB": {
        "propertyIndex": true, ...
}
```

17

Put them all into the index definition.
Put the property you want the results to sorted by as ordered true.

Optimize index definitions

adaptTo()

- Expand index definition by properties for special search features
  - Fulltext search
  - Facets
  - Excerpts
  - Suggestions

18

Start to optimize index definitions when queries run, use intended indexes but still
- are too slow or
- give unexpected results.

Expand index definition by properties for special search features needed like fulltext search, facets, excerpts and suggestions.

Fulltext search

- oak:index/myindex

```
"compatVersion": 2,
"type": "lucene",
"async": "fulltext-async",
"codec": "Lucene46",
"indexRules": { ...
... "properties": { ...
        "jcr:title": { ...
                "analyzed": true,
                "name": "jcr:title",
                "nodeScopeIndex": true,
                "propertyIndex": false,
                ...
```

Put all properties you want to search through, including those from subnodes of jcr:content

19

Fulltext search is of course a commonly used feature.
It needs a property set to analyzed = true like in this example of a lucene index definition.
Think carefully about which properties you want to search through, including those from deeper subnode of jcr:content.
Put them all into the index definition.

adaptTo()

Fulltext search

- Search for:
  `Grippeimpfstoff* -Ausschreibung "2016/2017"`
- Characteristics
  - AND Operation !
  - Wildcards *?
  - Negation -
  - Phrases

20

- What provides the fulltext search?
- Look at this example.
- The search term contains 2 words and a phrase
- It demonstrates that the fulltext search deals with wildcards, negation and phrases
- The most remarkable is, that the search term will be taken as AND Operation.
- That means the fulltext search will find documents where every single part of the search term matches.

adaptTo()

Facets

Put the properties from which you want to extract facets

```
...
"region": {
        "facets": true,
        "analyzed": true,
        "name": "region",
        "propertyIndex": true,
        ...
```

21

- Let‘s look at Facets.
- Put all properties into the index definition from which you want to extract facets and set facets = true.
- That is done easy.
- We have encountered in OOTB AEM 6.2 a problem setting facets=true with a multivalue property
- We could solve this problem by updating AEM with a fix pack which brought to our system a newer version of OAK
- Recommendation:
    - stay updated!
    - Keep your system updated.
    - Look at informations about new releases of updates an install them soon.

Facets result

adaptTo()

- Search result arranged into category, numerical count of how many matching documents

```
region :
 Berlin / Brandenburg (3)
 Hessen (4)
 keiner Region zugeordnet (6)
 Hamburg (3)
 Bundesweit (802)
 Rheinland-Pfalz (3)
 Baden-Wuerttemberg (5)
```


- Let‘s have a short look at what facets are.
- They arrange the search result into categories including a numerical count of how many matching documents found.

adaptTo()

Facets search

- Query-Debugger

```
0_property=region
```

- QueryBuilder API

```
Predicate p = new Predicate("region",
        JcrPropertyPredicateEvaluator.PROPERTY);
p.set(JcrPropertyPredicateEvaluator.OPERATION,
        JcrPropertyPredicateEvaluator.OP_EQUALS);
p.set(JcrPropertyPredicateEvaluator.VALUE, null);

query.getPredicates().add(p);
Map<String, Facet> m = query.getResult().getFacets();
```

23

- If prepared in index definition it‘s very easy to get the facets from the query along with the query results
- In Query-Debugger simply list the property beside the search criterias and mark a checkbox of the search formular.
- With QueryBuilder API simply add the predicate by name of the property to the query
- Then QueryBuilder will extract Facets for this property and you can get them very easily by calling a getter method.

Excerpts

adaptTo()

```
...
"jcr:title": { ...
        "analyzed": true,
        "name": "jcr:title",
        "nodeScopeIndex": true,
        "propertyIndex": false,
        "useInExcerpt": true, ...
},
"excerpt": { ...
        "notNullCheckEnabled": true,
        "propertyIndex": true,
        "name": "rep:excerpt", ...
}
...
```

Set special property rep:excerpt if you want to show excerpt's from query results

24

- Excerpts are a search feature that present a little part of the content of the search results, usually highlighting the search term.
- Recommendation:
    - Expand the index definition of all properties defined for fulltext search for use-in-excerpt as well
- Then set a special property called rep:excerpt to get excerpts faster.

Suggestions

```
"compatVersion": 2,
"type": "lucene",
"async": "async",
"suggestion": {
        "suggestAnalyzed": true,
        "suggestUpdateFrequencyMinutes": 480
},
"indexRules": { ...
... "properties": { ...
        "jcr:title": { ...
                "useInSuggest": true,
                "analyzed": true,
                "name": "jcr:title",
                "propertyIndex": true, ...
        }, ...
```

25

- A short look at suggestions
- My recommendation is:
- Define an separate index with all properties you want to obtain suggestions from.
- Very important is to define in the general part of the index definition „suggestAnalyzed“ = true, because otherwise whole property text would be presented in suggestions, what could be fairly long.
- Normally you want to have presented single words.

adaptTo()

Optimize queries

- Switch from XPATH to SQL-2
  - Avoid translation costs
  - Avoid too many OR-Conditions that lead to many union select statements
- Modify filter condition
  - Reduce the number of conditions
  - Find simpler conditions

26

The main reason for Optimization of queries is to improve the performance of a search.

- In OAK the default query language is SQL-2.
- All queries will be translated to and then executed as SQL-2 query.
- So one can optimize a query by switching from XPATH to SQL-2 as query language
  - One can avoid translation costs, because every xpath query will automatically translated into sql-2
  - Translation costs are not that high, but at least parsing and optimising are omitted
  - Because automatic translation from XPATH to SQL-2 generates for OR-Conditions many union select statements, one can avoid that by writing SQL-2 directly
- Another way for optimization is to modify filter conditions
  - One can reduce the number of conditions
  - Or find a simpler condition, for example the Exist-Condition is simpler than an equals Condition.

adaptTo()

XPATH to SQL-2 Translation: example

- XPATH

```
/jcr:root/content/siteOne//element(*, cq:Page)

[(jcr:like(jcr:content/@number,'prefixA%') or
jcr:like(jcr:content/@number,'prefixB%') or
jcr:like(jcr:content/@number,'prefixC%’))

and

(jcr:like(jcr:content/@thema,'%topicA%') or
jcr:like(jcr:content/@thema,'%topicB%') or
jcr:like(jcr:content/@thema,'%topicC%'))]
```


Let‘s look at an example for switching from xpath to sql-2
You have to find an correct statement expressing the same semantics.

XPATH to SQL-2 Translation: example

- SQL-2 generated

```
select [jcr:path], [jcr:score], * from [cq:Page] as a where
    isdescendantnode(a, '/content/siteOne’)
    and [jcr:content/number] like 'prefixA%’
    and [jcr:content/thema] like '%topicA%’
union select [jcr:path], [jcr:score], * from [cq:Page] as a where
    isdescendantnode(a, '/content/siteOne’)
    and [jcr:content/number] like 'prefixA%’
    and [jcr:content/thema] like '%topicB%’
union select [jcr:path], [jcr:score], * from [cq:Page] as a where
    isdescendantnode(a, '/content/siteOne’)
    and [jcr:content/number] like 'prefixB%’
    and [jcr:content/thema] like '%topicA%’
union select [jcr:path], [jcr:score], * from [cq:Page] as a where
    ... etc.
```

28

The automatic translation by the OAK implementation generates for this query 9 select/union select statements.

XPATH to SQL-2 Translation: example

- SQL-2 optimised („manually“)

```
select [jcr:path], [jcr:score], * from [cq:Page] as a where
    isdescendantnode(a, '/content/siteOne')

    and ([jcr:content/number] like 'prefixA%'
            or [jcr:content/number] like 'prefixB%'
            or [jcr:content/number] like 'prefixC%')
    and ([jcr:content/thema] like '%topicA%'
            or [jcr:content/thema] like '%topicB%'
            or [jcr:content/thema] like '%topicC%')
```

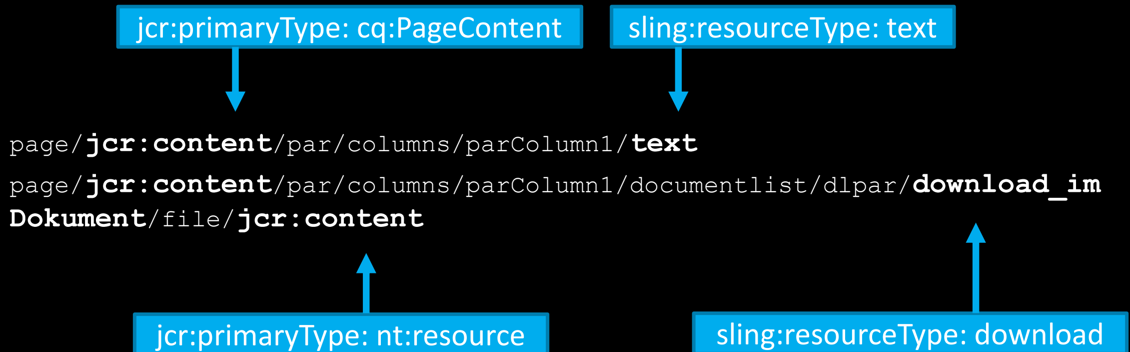

That is a manually optimised sql-2 query with the same semantics.
It leads to only one select statement which uses only once the optimised index.
So a great perfomance gain is achieved.

Modify filter condition: requirements

- Deep content structure inside a cq:Page
- Hierarchical mix of text, list and download
- Required results of type cq:PageContent and download
- Results ordered by jcr:score

30

Now an real life example for optimization by modifying filter condition
It came from our customer and had the following requirements:



The content could look like this.

- jcr:content with text properties
- A text node at level 4
- A download node at level 6
- It‘s a really deep content structure

text = mycomps/components/text
download = foundation/components/download
list = parsys

Modify filter condition: query

```
/jcr:root/content/siteOne//* [(jcr:contains(., 'query’))
and

(@jcr:primaryType = 'cq:PageContent’

or

@sling:resourceType = 'foundation/components/download’)]
```

32

- The query is a fulltext query
- Because of aggregation in index definition you find documents containing the text
  - In any text node and
  - In any download document in the substructure of cq:PageContent

BUT: query contains one OR-Condition that leads to 2 select statements combined with union select
and
The results will be sorted by the two nodetypes
At first come all results from type cq:PageContent and behind
All results from download type sorted by jcr:score

Modify filter condition: simplify condition

- Simple solution: add a further property to both nodes, that
  - Ideally already exists in one of the them
  - Ideally makes sense to show in result page
  - Is mandatory or automatically set by node creation
  - Is included in index definition

A simple solution is to add a further property to nodes of both types

Modify filter condition: modify content

```
page/jcr:content/par/columns/parColumn1/text/@text = "query"
page/jcr:content/@region = "Berlin"

page/jcr:content/par/columns/parColumn1/documentlist/dlpar/download_im
Dokument/@jcr:title = "query"

page/jcr:content/par/columns/parColumn1/documentlist/dlpar/download_im
Dokument/@region = "Berlin"
```

34

Add property @region to cq:PageContent and download

- Different Mechanisms possible
    - Mandatory field in edit control
    - Automatic initialization implemented by an EventHandler

Modify filter condition: simplify condition

```
/jcr:root/content/siteOne//* [(jcr:contains(., 'query’))
and

        @region
```

- and re-gain performance
  and sort order by jcr:score as well!

35

Customers performance re-gain – example:
Query1 – 161248 hits – 211 s vs. 54 s
Query2 – 25939 hits – 25,9 s vs. 6,4 s
Query3 – 19261 hits – 19,3 s vs. 3,4 s

Measuring search improvements

- By the old way: log-message
  - Compare log-message timestamps
- Take time output from Query-Debug-Log
  - Query time
  - Facet extraction time
- Test with Live Content !
- Bookkeeping in (excel) sheet

36

Let run the same queries after every single step of optimization and record logged time.
Select
- representative queries
- Queries with many and small results

- It‘s very crucial, that all performance test‘s are made at a system with Live Content or nearly the amount of the Live Content.
- Some of the performance problems are not visible and not evaluable at a devloper machine with small content.
- Every improvement must be validated under Live condition.

adaptTo()

Tools

- Query-Debug-Log
- Diagnosis-Tool
- Query-Debugger
- Last but not least CRXDE

37

Query-Debug-Log
- Log-Level Debug gives rich information about query transformation into sql2, index selection, predicates, index update etc.

Diagnosis-Tool
- Set of administration tools
- Beside other very useful tools there are Index Manager and Query Performance Tool
- „Explain Query Tool“ as part of Query Performance Tool is the most useful for understanding Query execution and query performance
- Look at http://localhost:4502/libs/granite/operations/content/diagnosis.html

Query-Debugger
- Favourite !
- Very fast, no overhead
- Shows all features needed
  facets, excerpts, pagination
- Based on QueryBuilder, therefore very useful for all developers implementing queries with QueryBuilder-API
- But useful as well for someone that wants to optimize his queries and proof the query results.
- Get used to it !

- Look at http://localhost:4502/libs/cq/search/content/querydebug.html

Last but not least CRXDE
- Disadvantages
    - Doesn‘t work with slow queries
    - Cannot show a lot of query results
- Good for searches for special values and quickly link you to the right data

Further information

- https://docs.adobe.com/docs/en/aem/6-3/deploy/platform/queries-and-indexing.html
- https://docs.adobe.com/docs/en/aem/6-3/administer/operations/operations-dashboard.html
- https://docs.adobe.com/docs/en/aem/6-3/develop/search/querybuilder-api.html
- https://hashimkhan.in/2015/12/02/query-builder/
- https://jackrabbit.apache.org/oak/docs/query/lucene.html
- http://www.aemstuff.com/blogs/feb/aemindexcheatsheat.html
- http://oakutils.appspot.com/

adaptTo()

Questions ???