Need for speed
How to use eZ Find search fetch instead of standard content list/tree fetch

By Ivo Lukač
http://www.netgen.hr/eng/blog

Index

Goal description

This tutorial should give eZ Publish developers some new ideas on how to use eZ Find. The ever growing need for speed can be achieved by using eZ Find search function instead of standard content list/tree fetch functions. Solr engine, used by eZ Find, is far more superior regarding read speed than mysql (at least in the way eZ Publish uses them). This fact becomes apparent when dealing with 10 or 100 thousands of objects in your database with complicated eZ Publish SQL queries starting to slow down rapidly. If you need text search capabilities difference in read speed becomes even more obvious.

Introduction

The main goal here is to replace standard content list/tree template fetch functions (which use SQL queries) with eZ Find search fetch function (which uses the Solr indexing engine). Reason for this is to gain more speed as Solr is way faster in case of large site with 10 thousands of objects or more. Additional benefit is the text searching capability of Solr, which can be used to enrich functionalities available on the website. There are some drawbacks and some situations where Solr search fetch function cannot be used for replacing standard eZ Publish fetch functions, and these will be covered.

This tutorial could be even more usable in the future because of the direction where eZ Find development is headed, as it will be possible to store entire objects in the Solr index. In this case it will not be needed to use database at all, in order to fetch content nodes.

Today, although we cannot avoid using SQL queries entirely (we can get list of nodes from Solr but the node content is still fetched from the database), performance gains in using Solr instead of standard eZ Publish fetch functions can be huge.

Pre-requisites and target population

This tutorial is written for experienced eZ developers who already use eZ Find, as well as for intermediate eZ developers who did not yet use eZ Find but are planning to do so. As we will not cover installation of eZ Find, the main requirement is to have eZ Find installed and working. More information on eZ Find can be found here: http://ez.no/ezfind

Step 1: Understanding the basic notions

Before digging in it is important to know few things.

The first thing you need to be aware of is that eZ Find is using database of its own, based on the well known Solr/Lucene search engine. More info on Solr can be found here: http://lucene.apache.org/solr/. A citation from that web page:

"blazing fast open source enterprise search platform"

So the content needs to be indexed (transferred to Solr database) every time it is created or changed. If for some reason the indexation process fails you will not have up-to-date data in the index.

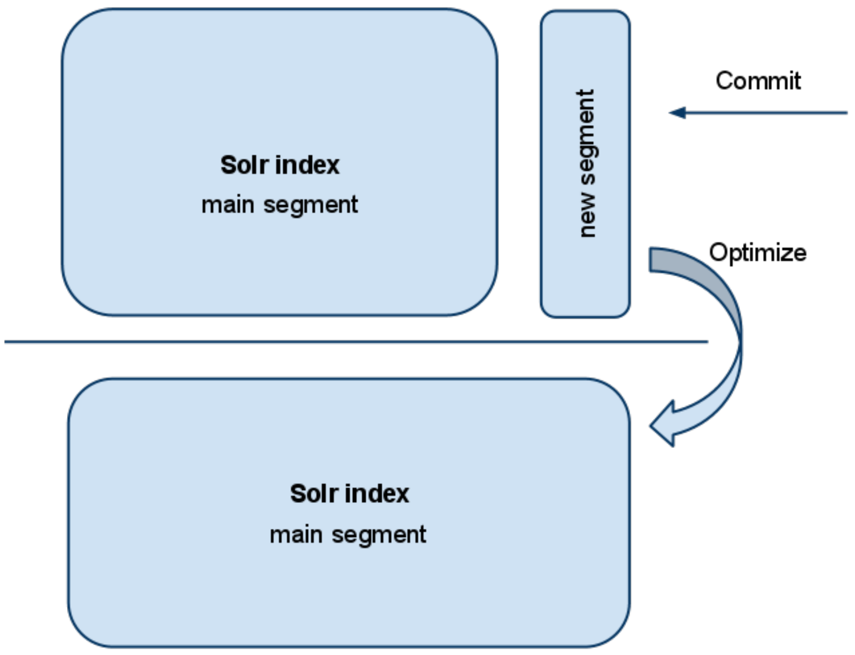
The indexation task is carried out by eZ Find search plugin. There are 3 ways how it can be configured:

- after publish (drawback is that it burdens the publish process),
- delayed to cronjob (drawback is that the index is always lagging behind so it may happen that query results are not 100% correct)
- a combination of first two (new in eZ Find 2.2) with DisableDirectCommits option. Publish is faster as indexing is queued within Solr and not done immediately (process is commited every CommitWithin seconds)

There is no silver bullet solution in choosing from these 3 options, as it depends on the way you are using search functions. If results need to be up-to-date then DelayedIndexing option in ezfind.ini should be disabled. In that case for faster publish DisableDirectCommits and CommitWithin could be used.

Good practice would be to launch complete reindexing every week just to be sure, because various things can go wrong here: external tools for binary files can break, etc.

Second important thing is the 'optimize' function. Optimize does exactly what the name suggest, basically it merges more Solr segments (created by update, delete, etc.) into one and by doing so makes searching faster. Recommendation would be to schedule this with cronjob as it's not important to be executed right after content is changed. Therefore OptimizeOnCommit option in ezfind.ini should be disabled.



Third thing is to enable AllowEmptySearch in site.ini. It will enable use of eZ Find search function for listing nodes without having any search text query. This switch is generally useful to remain disabled if standard eZSearch engine is used (to prevent exhaustive SQL queries) but with eZ Find it is not an issue, as Solr handles this much better.

Fourth: indexed fields are only subset of all fields from the database. What fields you can use depends on what meta data eZ Find maps, what class attributes are searchable and what object attributes are marked searchable.

Fifth, and last, with version 2.2 eZ Find you have the option to segment the index into more chunks which can be configured independently – called shards. This can be useful for many things, among which are:

- multilanguage site to distinguish language dependent settings: collation, spellchecker, stemming, etc.
- search other indexes

More on shards here:
http://ez.no/doc/extensions/ez_find/2_2/advanced_configuration/using_multi_core_features

Sixth: Using eZ Find will give you performance boost but this should not prevent you to use all other caching possibilities of eZ Publish. To have a page with minimum request to SQL database or Solr engine is still a must.

Recommended settings

To conclude this basic options here are recommended settings in settings/override/site.ini.append.php:

```
[SearchSettings]
DelayedIndexing=disabled
AllowEmptySearch=enabled
```

In extension/ezfind/settings/ezfind.ini:

```
[IndexOptions]
OptimizeOnCommit=disabled
DisableDirectCommits=true
CommitWithin=2
```

Step 2: Understanding eZ Find's fetch function for search

In fact, the eZ Find search function is still using database but only for fetching node data after search result list is returned by Solr. SQL queries for getting node data are rather fast and do not present a real problem. Results are dependent on user rights so we don't need to worry about access privileges also.

Main characteristics of the search template function:

- It can search text with in all indexed fields (query parameter)
- There are offset and limit parameters for paging (offset & limit parameters)
- There is a sort parameter (sort_by parameter). Default sort is relevance based, but other sorting can be used also.
- Facets enable drill-down possibilities (facet parameter)
- Filtering with multi nesting conditions on all attributes and meta data (filter parameter)
- Special class parameter for filtering more classes (class_id parameter)
- Subtree array parameter for filtering one or more subtrees (subtree_array parameter)
- Special section parameter for filtering on sections (section_id parameter)
- Ignore visibility parameter to disable or enable of searching hidden nodes (ignore_visibility parameter)
- Limitation parameter for overriding current user access limits (limitation parameter)

There is no parameter for fetching list (instead of tree) but it can be easily achieved with filter. It is possible to search within more than one parent node. Results can be sorted by relevance, by meta-data or by attributes. Filter can be built with mix of nested "AND" and "OR" conditions.

The most important gain, if the eZ Find search function is used, is the possibility to combine filtering with powerful text search. And there are lot of bonus features that can be used also: highlighting , spellchecking, etc

Step 3: Mapping legacy fetch parameters to eZ Find search parameters

content list/tree function parameters	eZ Find search function equivalent	
	Tree	List
`Tree or list`	By default	Add to filter following condition: main_parent_node_id:[PID]
`parent_node_id`	`subtree_array`	
`sort_by`	`sort_by` More or less the same except: - Default is score/relevance ranking based on boost parameters - No priority sort - No depth sort - path_string should be sortable by specifying url_alias field	
`offset & limit`	`offset & limit`	
`attribute_filter`	`filter` In standard content list/tree function only 1 type of condition can be put (AND or OR). eZ Find supports more conditions and these conditions can be nested. Keep in mind that matching is different. E.g. 'in' can be replaced with (term1 OR term 2). Possibilities to use: - combinations: 'OR' and 'AND' with nesting - ranges: [0 TO *], [* TO 5] , [10 TO 20] - obligatory ('+') and negative ('-') operators	
`extended_attribute_filter`	`filter` or eZ Find special rawSolrRequest function Solr query possibilities are different from SQL query possibilities so direct comparison does not make much sense. eZ Find rawSolrRequest function can use only data stored in the index so it will be less capable then extended_attribute_filter which is written in PHP and can use all	

	data from the database.
class_filter_type & class_filter_array	**class_id** For including classes. Excluding can be done through ezfind.ini ([IndexExclude] section)
only_translated & language	**filter** To narrow down to specific language results can be filtered with e.g. language_code:ger-DE There is also a SearchMainLanguageOnly switch in [LanguageSearch] section in ezfind.ini for using only prime language. Otherwise SiteLanguageList[] setting in site.ini is used. For leveraging even more from Solr shards can be used as a specific index for every language. In that case specific Solr configuration can be applied per language e.g. collation, stemming, etc.
main_node_only	No data in the index. Generally always returns the main node, but finds the object in other locations also.
as_object	Not implemented yet.
depth	No data in index.
limitation	limitation
ignore_visibility	ignore_visibility

Step 4: Examples

Few simple examples on how to replace fetch content calls with eZ Find calls.

Example 1

Using standard function:

```
{fetch( 'content', 'list',
     hash( 'parent_node_id', 100,
           'class_filter_type', 'include',
           'class_filter_array, array('article'),
           'sort_by', array( array('modified',false()),
                             array( 'attribute', true(),'article/title') )
))}
```

Using eZ Find:

```
{fetch( 'ezfind', 'search',
     hash( 'filter', 'main_parent_node_id:100',
           'class_id', array('article'),
           'sort_by', hash( 'modified', 'desc', 'article/title', 'asc' ) ) )}
```

Listing only child nodes is solved with special filter. Including classes is simpler. Sorting is a bit different and it needs only one hash, with no nested arrays.

Example 2

Using standard function:

```
{fetch( 'content', 'tree',
        hash( 'parent_node_id',     100,
              'ignore_visibility', true(),
              'limit', 20,
              'offset', 0,
              'attribute_filter', array( array( 'review/rating',
                                                'between',
                                                array( 0, 2 ) ) ) ) )}
```

Using eZ Find:

```
{fetch( 'ezfind', 'search',
        hash( 'subtree_array', 100,
              'ignore_visibility', true(),
              'limit', 20,
              'offset', 0,
              'filter', array('review/rating:[0 TO 2]') ) )}
```

This example shows even more similarity between standard fetch function and eZ Find search fetch function. Only difference is the filter with the way how condition is constructed.

Step 5: Leveraging what is beyond the standard functionality

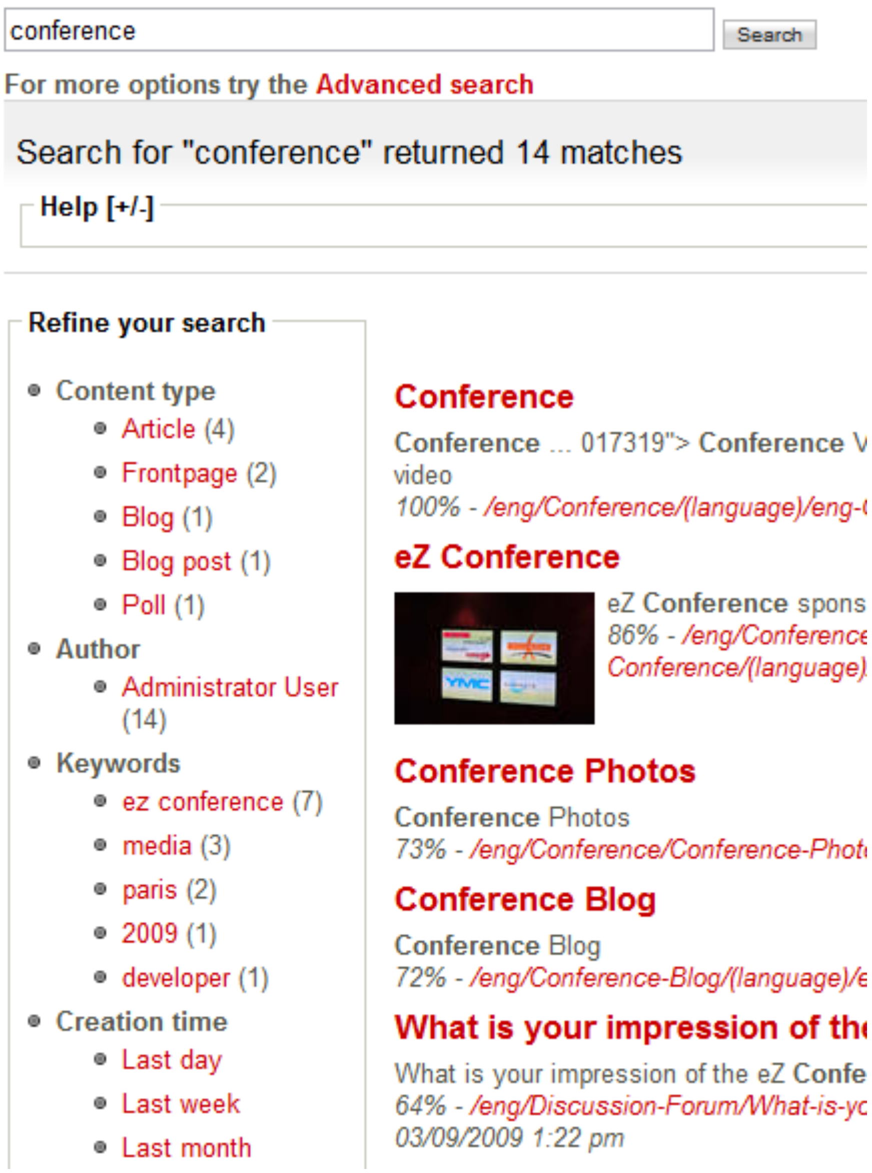
A few features that standard content fetch functions do not have and eZ Find does (and that are rather usable):

Text search

Text search is the most powerful feature you can use. Searching for more words can be configured with "AND" or "OR" logic. Special signs "+" and "-" are used for defining obligatory and negative search terms if there are more of them. Quoting more words searches for exact phrases.

Facets

Facets are tools for drilling down within fetch results. There are extremely useful for giving user more information about the data presented by showing number of nodes per facet and by giving possibility to refine results with clicks. All information about facets is returned within the same result set so only one fetch is needed.


conference

Search

For more options try the Advanced search

Search for "conference" returned 14 matches

Help [+/-]

Refine your search

- Content type
  - Article (4)
  - Frontpage (2)
  - Blog (1)
  - Blog post (1)
  - Poll (1)
- Author
  - Administrator User (14)
- Keywords
  - ez conference (7)
  - media (3)
  - paris (2)
  - 2009 (1)
  - developer (1)
- Creation time
  - Last day
  - Last week
  - Last month

Conference

Conference ... 017319"> Conference V
video
100% - /eng/Conference/(language)/eng-(

eZ Conference

eZ Conference spons
86% - /eng/Conference
Conference/(language)

Conference Photos

Conference Photos
73% - /eng/Conference/Conference-Phot

Conference Blog

Conference Blog
72% - /eng/Conference-Blog/(language)/e

What is your impression of the

What is your impression of the eZ Confe
64% - /eng/Discussion-Forum/What-is-yo
03/09/2009 1:22 pm

More info:
http://ez.no/doc/extensions/ez_find/2_2/customization/customizing_facets_and_drill_down_navigation

Boosting

Boosting is usable only when sorting by relevance. Can be:

- defined ad-hoc in query, e.g. attr_name_t^1.5
- configured within ezfind.ini [IndexBoost] section based on meta fields, class, attributes, etc

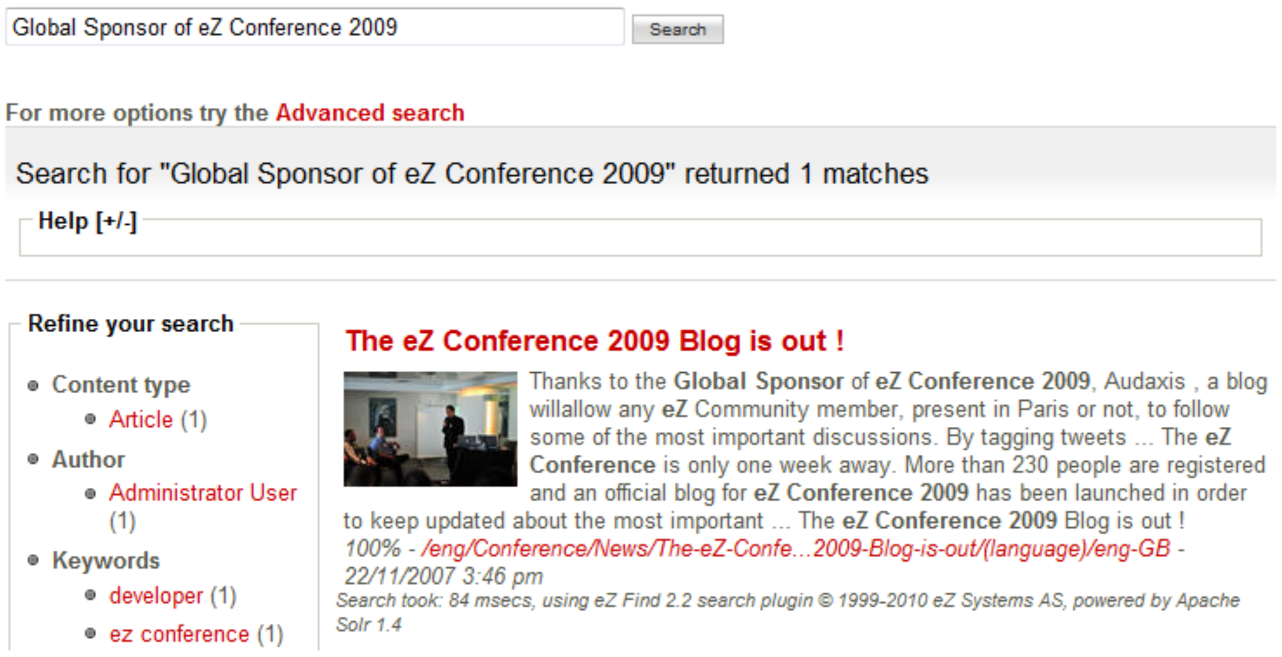
More on boosting:

http://ez.no/doc/extensions/ez_find/2_2/advanced_configuration/index_time_boosting

http://ez.no/doc/extensions/ez_find/2_2/use/advanced_search/tunable_relevancy_ranking

Highlighting

Highlighting is usable only when searching for text. Can emphasize search terms in context where they appear.



Global Sponsor of eZ Conference 2009 Search

For more options try the Advanced search

Search for "Global Sponsor of eZ Conference 2009" returned 1 matches

Help [+/-]

Refine your search

- Content type
  - Article (1)
- Author
  - Administrator User (1)
- Keywords
  - developer (1)
  - ez conference (1)

The eZ Conference 2009 Blog is out !

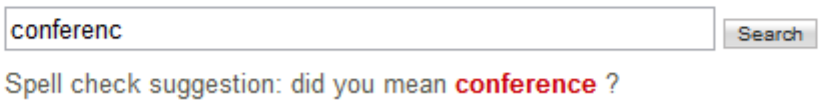
Thanks to the **Global Sponsor** of **eZ Conference 2009**, Audaxis , a blog willallow any **eZ** Community member, present in Paris or not, to follow some of the most important discussions. By tagging tweets ... The **eZ Conference** is only one week away. More than 230 people are registered and an official blog for **eZ Conference 2009** has been launched in order to keep updated about the most important ... The **eZ Conference 2009** Blog is out !

100% - /eng/Conference/News/The-eZ-Confe...2009-Blog-is-out/(language)/eng-GB - 22/11/2007 3:46 pm

Search took: 84 msecs, using eZ Find 2.2 search plugin © 1999-2010 eZ Systems AS, powered by Apache Solr 1.4

Spell checking

Spell check is also usable only when searching for text. Can show suggestions for corrected terms based on indexed values.



conferenc Search

Spell check suggestion: did you mean **conference** ?

More info: http://ez.no/doc/extensions/ez_find/2_2/use/advanced_search/spellchecking

Step 6: How to ponder the replacement of a legacy fetch by eZ Find's search

Benefits:

- Speed, speed & speed
- Score ranking
- Native functionalities: search, facets, boosting, elevation, highlighting, etc.
- Shards for multi language sites
- No need for extra count function
- Simpler and more powerful filtering
- Speed

Drawbacks:

- Data duplication heavily dependent on syncing to be up-to-date
- No sort by priority nor depth (could be easily implemented by storing this information as meta fields)
- No main_node_only switch
- No class excluding in query, just an ini setting
- Filtering can use only data in the index (which is a subset of data from the database) and there is no way of defining extended filter in PHP

Some ideas for eZ Find improvements:

- Increase robustness for indexing to be sure that all data is indexed
- Index more meta data like depth, priority, main node bool, etc.
- Directly index binary files in Solr

Conclusion

If you have large site with lot of object and lot of page views performance is often a very important issue. View caching, cache blocks, static cache and reverse proxies are, of course, important ways to gain performance. But there are situations where the raw fetch speed is also very important. If you can manage to implement those fetches with eZ Find you will have instant positive effects:

- fetch itself is faster,
- database has less SQL queries to process and therefore can faster deal with other (concurrent) queries.

Additional functionalities that eZ Find provides are like a cream on top.

Happy coding!

Resources

http://ez.no/doc/ez_publish/technical_manual/4_x/reference/modules/content/fetch_functions/list

http://ez.no/doc/extensions/ez_find/2_2/customization/template_fetch_functions

About the author : Ivo Lukač

Working at Netgen, Zagreb, Croatia
Tech: CMS, eZ Publish, eZ Find, PHP, Linux, MySQL, Apache, Varnish
Services: system architecture, consulting, development, support, maintenance, upgrades

License choice