ISLG_MVC Project

General Notes

DtSearch IndexCache is not utilized. Consider initializing IndexCache on the application level and passing it to SearchJob instances. See more at:
https://support.dtsearch.com/webhelp/dtSearchNetApi2/dtSearch__Engine__IndexCache.html
An index cache can make searching substantially faster when a series of searches must be executed against a small number of indexes. The index cache maintains a pool of open indexes that will be available for searching on any thread. This eliminates the need to open and close the indexes being searched for every search request.

DTSearchController.cs

DTSearchFTS() - Full Text Search handler

1) Serializes the complete result list, just to parse it again, as in:

```
var lists = res.SerializeAsXml();
XDocument resdocs = XDocument.Parse(lists);
```

2) For each query there's an SQL statement created and executed. However, application code that uses results variable "rsGetTerms" is commented out. Unless this code removal was temporary, SQL executing could be removed as well.

3) The major bottleneck slowing down the full text search is invocation to *ReturnDocsByFilter()*. This is critical as can be seen in the below benchmark results.

4) For each document in results, there's a call to *res.MakePdfWebHighlightFile()* to create PDF highlight file, which is then saved in user session. Although generally fast, this creates an additional pressure on CPU and memory usage. It could be refactored to create highlight file only when PDF open is actually requested.

5) There's an extensive use of string concatenation of variable "dispContent". It's recommended to use to StringBuffer instead.
https://stackoverflow.com/questions/73883/string-vs-stringbuilder

Benchmark for "article" keyword search gives the following execution times:

- dtSearch search execution: 62 ms
- Serialize and deserialize results: 707 ms

- ReturnDocsByFilter: 29579 ms
- For each returned document:
    - MakePdfWebHighlightFile: 15-60 ms
    - About 100-150 ms on everything else - still a lot considering it should be only creating an output.

AdminFTS.cs

AdminDTSearchFTS()

Very similar to the *DTSearchFTS()* - the same comments apply.

PartialViewsController.cs

dsp_keywordSearch()
Uses hybrid approach using dtSearch for full text search, passing results as an input to SQL query to get additional fields.
If documents in dtSearch index and SQL records are in 1:1 relation and data in SQL doesn't change often, it should be considered adding of needed SQL columns to dtSearch index and avoid SQL querying.
Recommended doing a benchmark first to test how much SQL execution affects the actual response time.

UserControllerSearch.cs

GetSubjectNavigatorGlobalSearch()
Uses the same hybrid search approach as *dsp_keywordSearch()* - the same comments apply. (It looks like that here only the count of SQL results is used so maybe at least SQL could be modified to return count only instead of data. Again, should be checked how it affects response time.)

FullTextGlobalSearch()
Simple dtSearch search, used to return number of matching documents. Looks fine.

Other Searches

It appears that search in *Treaties & Rules* and *Dispute Documents* uses SQL to search for names/titles of documents that might be already present in the full text index, or at least could be indexed with a custom indexer.

DisputeDocumentGlobalSearch()
Invokes db stored proc DisputeDocumentNewGlobalSearch() to search across four columns using SQL 'like'.

TreatiesandRulesGlobalSearch()
Calls TreatiesController.Search() which invokes db stored proc usp_GetAnnotTreeBySearchAnnot()

Jurisprudence Citator – auto-complete
getalldocsrefs() in PartialViewsController.cs
Based on SQL substring search. Consider indexing with dtSearch.

Article Citator - auto complete
getfilterarticles() in PartialViewsController.cs
Invokes stored proc usp_getfiltercitator()

Indexing Process

It appears that Full Text Search and Treaties & Rules and Dispute related to the same documents set. Instead of using dtSearch Desktop for document indexing, consider creating a custom dtSearch indexer that would combine document metadata (used by Treaties & Rules and Dispute) and full text into a single index.
In addition, as ReturnDocsByFilter function is a huge performance bottleneck, try to eliminate it by adding all needed metadata data to the index.

To do this, you need to extend dtSearch.Engine.DataSource interface. (Several examples shipped with dtSearch Developer distribution demonstrate this.)
It's possible to simply add custom metadata to indexed document (e.g. from a database), reference file and leave document indexing to dtSearch. See https://support.dtsearch.com/webhelp/dtSearchNetStdApi/dtSearch_Engine_DataSource_DocIsFile.html

To speed-up other document searches and auto-complete functions, consider building dtSearch index from SQL data and use dtSearch instead of SQL. dtSearch Developer distribution includes some code samples that can be used as a base.
http://support.dtsearch.com/dts0111.htm