

# Re: Fwd: NUCULAR fielded text searchable indexing

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2007-10/msg01334.html>

---

- *From:* [aaron.watters@xxxxxxxx](mailto:aaron.watters@xxxxxxxx)
  - *Date:* Thu, 11 Oct 2007 13:19:05 -0000
- 

On Oct 11, 3:30 am, Paul Rubin <<http://phr...@xxxxxxxxxxxxxxxx>> wrote:

"Delaney, Timothy (Tim)" <[tdela...@xxxxxxxx](mailto:tdela...@xxxxxxxx)> writes:  
As for [ <http://nucular.sourceforge.net> ]  
itself, I've only had a chance to glance at the  
site...

Thanks for taking a look. I hope you don't mind if I disagree  
with a number of your comments below.

...but it looks a little more akin to Solr than to Lucene. Solr is  
a Java application that wraps Lucene to present a more convenient  
interface

I'm not sure but I think nucular has aspects of both since  
it implements both the search engine itself and also provides  
XML and HTTP interfaces, like

<http://www.xfeedme.com/nucular/gut.py/go>

...although it does this using XML over an HTTP socket. One  
thing about Solr and probably anything else like this: if you want to  
handle a high query load with big result sets, you absolutely must  
have enough ram to cache your entire index. ...

Of course it would be nice to have everything in RAM, but  
it is not absolutely necessary. In fact you can get very good  
performance with disk based indices, especially when the index  
is "warm" and the most heavily hit disk buffers have been  
cached in RAM by the file system.

As a test I built an index with 10's of millions of entries  
using nucular and most queries through CGI processes clocked

Re: Fwd: NUCULAR fielded text searchable indexing

in in 100's of milliseconds or better — which is quite acceptable,  
for many purposes.

...Since ram is expensive  
and most of it will be sitting there doing nothing during the  
execution of any query, it follows that you want multiple CPU's  
sharing the ram. So we're back to the perennial topic of parallelism  
in Python...

....Which is not such a big problem if you rely on disk caching  
to provide the RAM access and use multiple processes to access  
the indices.

The global interpreter lock (GIL) can be kinda nice sometimes too.  
When I built the big indices mentioned above it took a long  
time. If I had used a multithreaded build my workstation  
would have been worthless during this time.  
As it was I was using Python in  
a single process, so the other CPU was happy to work with me  
during the build, and I hardly noticed any performance  
degradation.

In one java shop I worked in the web interface worked  
ok (not great) provided it was only hit by one access at a time,  
because the system was engineered so that one access  
consumed all computational resources on the box. Of course  
if you hit it continually with 4 concurrent accesses the  
system went to its knees and got tangled up in livelocks  
and other messes. In the case of web apps it can be  
very nice to have one access handled by one process in one  
thread leaving the other cpu's available to handle other  
accesses or do other things.

imho, fwiw, ymmv, rsn, afaik.  
— Aaron Watters

===  
even in a perfect world  
where everyone is equal  
i'd still own the film rights  
and be working on the sequel  
— Elvis Costello "Every day I write the book"

.