

## Re: faster data structure to count word frequency?

*Source:* <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2005-03/2205.html>

---

*From:* Chris Uppal ([chris.uppal\\_at\\_metagnostic.REMOVE-THIS.org](mailto:chris.uppal_at_metagnostic.REMOVE-THIS.org))

*Date:* 03/21/05

Date: Mon, 21 Mar 2005 10:36:51 -0000

Kevin wrote:

> *Suppose we are required to count the frequency of each different word*  
> *in a large text documents,*

Since you mention that you are dealing with large documents, I will assume that you are dealing with 100s of megabytes (since otherwise there'd be little point in the kinds of optimisation you are considering).

So, some thoughts. Note that none of these are backed by measurements, they are just suggestions of things you might want /to/ measure.

First off. The disk read time for large (collections of) documents is significant. Quite possibly more so than all the processing you will subsequently do. If that is the case, then the only way to speed up the processing is to reduce the amount of data you read (e.g. you could try keeping the data on-disk in compressed form), or buy faster disks. BTW, when you come to measure this, remember that the OS will cache small (10s of megabytes) files pretty effectively, so its hard to get a feel for what the real disk IO time is; and impossible to measure /anything/ by looping reading the same small file several times.

Second. In the old days, file IO was often dominated (after the disk IO times) by the time it took to copy the data around in program-space. There were several versions of 'C' <stdio> that provided real gains by avoiding that overhead (I wrote one myself). I don't know to what extent such considerations are still relevant on modern machines (they'll definitely be /less/ relevant).

Third. Your main loop (once you've got the data into memory) is to

```
create a string (two object creates plus a copy [*])
do a hash table lookup
do the addition
do a second, identical, hash table lookup
```

which leaves considerable scope for optimisation since most of that work is wasted. You could create a special kind of hash table that could hash and find a substring of an existing string (or a subsequence of a char array), thus

comp.lang.java.programmer: Re: faster data structure to count word frequency?

avoiding the expense of creating 10s of millions of String objects that would (mostly) be immediately discarded (assuming that most strings would turn out to be in the hash table already). You would ensure that the values in the hash table were 'int's (or maybe 'long's), since if you are creating your own hash table it would be a significant waste of time and space (/and/ add unnecessary complexity) to mess around with Integer values. You would include in that hash table implementation an operation like incrementCountAt(), so that you could avoid the double lookup.

All of that would be moderately complicated to implement (although it would also be moderately interesting), so before you do that you should:

- create a test collection of representatively large documents
- create a test program that counts the words using the simplest approach.
- time it.

Then comment out the bit that actually counts the words (leaving just the IO), and time the cut-down version.

How much difference is there ? If it's small (as it might be), then you'd just be wasting your time trying to optimise the word counting.

-- chris

([\*] though the buffer-sharing that Strings do needs to be taken into account here too)