The Old New Thing

The cost of trying too hard: Splay trees

18 Jan 2006 10:00 AM | 22

Often, it doesn't pay off to be too clever. Back in the 1980's, I'm told the file system group was working out what in-memory data structures to use to represent the contents of a directory so that looking up a file by name was fast. One of the experiments they tried was the splay tree. Splay trees were developed in 1985 by Sleator and Tarjan as a form of self-rebalancing tree that provides $O(\log n)$ amortized cost for locating an item in the tree, where n is the number of items in the tree. (Amortized costing means roughly that the cost of M operations is $O(M \log n)$. The cost of an individual operation is $O(\log n)$ on average, but an individual operation can be very expensive as long as it's made-up for by previous operations that came in "under budget".)

If you're familiar with splay trees you may already see what's about to happen.

A very common operation in a directory is enumerating and opening every file in it, say, because you're performing a content search through all the files in the directory or because you're building a preview window. Unfortunately, when you sequentially access all the elements in a splay tree in order, this leaves the tree totally unbalanced. If you enumerate all the files in the directory and open each one, the result is a linear linked list sorted in reverse order. Locating the first file in the directory becomes an $O(n)$ operation.

From a purely algorithmic analysis point of view, the $O(n)$ behavior of that file open operation is not a point of concern. After all, in order to get to this point, you had to perform n operations to begin with, so that very expensive operation was already "paid for" by the large number of earlier operations. However, in practice, people don't like it when the cost of an operation varies so widely from use to use. If you arrive at a client's office five minutes early for a month and then show up 90 minutes late one day, your explanation of "Well, I was early for so much, I'm actually still ahead of schedule according to amortized costing," your client will probably not be very impressed.

The moral of the story: Sometimes trying too hard doesn't work.

(Postscript: Yes, there have been recent research results that soften the worst-case single-operation whammy of splay trees, but these results weren't available in the 1980's. Also, remember that consistency in access time is important.)

Blog - Comment List MSDN TechNet

Comments

denis bider
18 Jan 2006 10:18 AM
#
This isn't about trying too hard, it's about trying the wrong way. Trying hard isn't bad, trying the wrong way is.

Steve Bjorg
18 Jan 2006 10:58 AM

#

Consistent delivery of results is a the heart of six-sigma. Maybe there is am opportunity here to research algorithms that perform well and are highly predictable in their performance, not just amortized. Or, alternatively, provide a measure of 'consistency' of performance for a problem of size 'N'. Just a thought.

Joe Beda

18 Jan 2006 11:58 AM

#

Hey Raymond, I can't go into details, but I *know* that there are other projects at Microsoft that used splay trees to great affect -- so don't give them too bad a rep.

But I agree with your thesis -- understand all of the consequences of your algorithms and use the simplest one that will solve your problem.

Perhaps there is some sort of twist on occams razor: "When presented with multiple ways of doing something, do the simplest." :)

anon

18 Jan 2006 11:58 AM

#

Raymond, I don't know if I have isolated properly, but clicking on folders using Windows Explorer sometimes causes the "desktop icons explosion". Have you posted anything about that yet? And btw, is there a fix?

AC

18 Jan 2006 12:40 PM

#

Does anybody know why directory operations with NTFS are so slow on Windows when there are a lot of files in the directory (e.g. IE cache files)? I have some impression that directories are kept extremely fragmented and too spread around (And I admit I don't know if it is fixed in the newest and latest incarnations).

Anybody knows more on the subject?

Chris Peterson

18 Jan 2006 1:45 PM

#

An old-timer once told me that computer science might have many fancy-pants data structures, real-world software development only has three: stack, queue, and hashtable. :)

KJK::Hyperion

18 Jan 2006 1:45 PM

#

AC: Explorer isn't a good filesystem benchmark. In most cases, even displaying the list of files implies opening every single file and at least reading its attributes (at worst even some data). And remember that every instance of the shell is single-threaded and very little happens asynchronously. The "dir" command is a better indicator of directory enumeration speed

thosk
18 Jan 2006 2:19 PM
#
NTFS perf involving folders with "many" files can also be impacted if the files do not conform to 8.3 convention *and* automatic 8.3 names have been created for the files.

Anon
18 Jan 2006 3:49 PM
#
Perhaps the team should focus more on on-disk stuff than on-memory stuff.

On-memory, Windows is quite fast, no problem there. On-disk, well, NTFS is the slowest thing you'll find (except for directories will *lots* of files, where it can beat FAT, though it is held down by 8.3 conversions unless you have disabled them), followed by FAT, and then pretty much everything else.

In Vista, they have a lot of focus on "covering" this with more intelligent prefetching, which is good, but would be even better if NTFS was improved; and since I've heard no mention of it, I can't have high hopes...

I hate to compare but I find this rather funny: in low-end machines, when Windows runs slow is mostly always because of I/O: you can hear the hard disk trashing badly (and it's even more frustrating because of the noise). When linux runs slow, it's mostly because the way it's designed, with the X server running as another process and the protocol-based communication. It's eerie to watch it run slowly, but not doing any disk I/O.

Moz
18 Jan 2006 4:27 PM
#
> Explorer isn't a good ... even displaying the list of files implies opening every single file

This is something that annoys me. I'd love it if there was a way to turn the behaviour off, the way a bit of registry hacking can turn off the "search inside zip files" feature.

Robert
18 Jan 2006 6:41 PM
#
> The "dir" command is a better indicator of directory enumeration speed

Actually, the "dir" command spends most of its time scrolling and updating the console window.

Yuliy
18 Jan 2006 7:49 PM
#
>Actually, the "dir" command spends most of its time scrolling and updating the console window.

dir > file

Vince P
18 Jan 2006 11:37 PM
#
I know this is off-topic. but someone mentioned something about having Explorer browser a folder with a ton of files, and another person mentioned about how slow Windows gets with I/O

Can I say AMEN?

I wish computers/mouses came with shock absortion detection so that the O/S can be informed of how infuriated I'm getting waiting for it to do something that turned out to be not worth the wait.

Why can't Task Manager have super-duper (to borrow language from the Alito hearings) Thread priority so that the CPU can cool its heels and give me enough cycles to kill something I want to die.

When Windows starts collapsing under its own virtual page file weight it's the most annoying and aggravating experience. (Yes, I know CPU and Page File are not the same thing but either way, it prevents me from getting on with it) I get where i want to take my laptop and fling it at someone.

Chris Peterson
19 Jan 2006 12:59 AM
#
Vince: I recently discovered a shareware utility called "Process Tamer". It runs in the background and changes processes' priorities based on YOUR choice. So I set iTunes.exe and Firefox.exe to AboveNormal priority and my compiler and linker to Low priority, so I can still browse the web while compiling. And if any process runs at 100% CPU for N seconds, Process Tamer can proactively slam it to Low priority so it doesn't lock up your PC. Why doesn't Windows already do this?!

http://www.donationcoder.com/Software/Mouser/proctamer/

Norman Diamond
19 Jan 2006 2:11 AM
#
I seem to be missing something here.

> Splay trees were developed in 1985 by
> Sleator and Tarjan as a form of self-
> rebalancing tree that provides O(log n)

> amortized cost for locating an item in the
> tree,

OK. And surely rebalancing is done each time an item is inserted, right?

> Unfortunately, when you sequentially access
> all the elements in a splay tree in order,
> this leaves the tree totally unbalanced.

Do you mean that anti-rebalancing is done each time an item is fetched? If so, might you happen to have a reference to a statement by Sleator or Tarjan saying why they did it?

I've seen a few algorithms that reorder array elements on fetches when the structure is an array with no links[*], but other structures such as binary trees, threaded binary trees, hash tables, etc. were rebalanced when elements were inserted.

Even if rebalancing was performed when elements were fetched, if it was a rebalancing operation then it wouldn't straighten out the entire structure, it would balance the structure. Threading links would remain in place for anyone who wanted to do a sequential retrieval, but a random search wouldn't use them. What kind of anti-rebalancing operation, I just can't imagine.

By the way balanced binary trees work better for structures in RAM than on disk. They even work better for structures that stay in RAM than for structures that get paged out to the paging file. Maybe if I try hard then I might be able to imagine why splay trees would have been invented with this kind of behaviour. But more likely I think I'm just missing something with regard to how they were really designed.

[* Designed at a time when the average mainframe didn't even have the amount of memory that Windows 1.0 would want to run in, but I kind of doubt whether it was ever worth while to avoid storing a reasonable amount of links.]

Nick Lamb
19 Jan 2006 6:07 AM
#
"it's mostly because the way it's designed, with the X server running as another process and the protocol-based communication."

Did you actually /measure/ this effect, or are you, like so many before you, /assuming/ that you're smarter than the people who designed X11 and thus don't need to do your homework ?

Hint: The pixels only go through the protocol socket when destined for a remote server.

Anon
19 Jan 2006 8:39 AM
#
Nick Lamb: That's off-topic here, but anyway:

I know on localhost it uses "more efficient transport means", but I always assumed that transport method aside, the X11 protocol is always used, and you have to encode/decode from it, on two separate processes.

Anyway, on a P3-800 Mhz, Windows is very fast as far as drawing windows is concerned, even with XP themes. Whereas linux is quite slower, some X11 apps can take seconds to start even from disk cache and screen redraws are noticeably slow. I don't know who the culprit is, but I think it's X11 on the basis that stuff like video playback or OpenGL games do run as fast as in Windows.

M Knight
19 Jan 2006 8:41 AM
#
> AC: Explorer isn't a good filesystem benchmark. In most cases, even displaying the list of files implies opening every single file and at least reading its attributes (at worst even some data).

I've read somewhere it takes about 31 I/O operations per file to display a file in the Explorer GUI

Per Vognsen
19 Jan 2006 7:04 PM
#
>OK. And surely rebalancing is done each time an item is inserted, right?

Err, no.

There is no balance invariant a la AVL or red-black trees, no. Every time an element is accessed in a splay tree, the corresponding node is rotated step by step to the root. It is exceedingly easy to prove that doing this to the elements in sorted order yields an entirely lop-sided tree; either left or right lop-sided, depending on whether the elements were in increasing or decreasing sorted order.

ReuvenLax
19 Jan 2006 8:19 PM
#
Yeah, Dan Lovinger told me that story a couple of years ago. This is the cost of not understanding your tools. Splay trees (actually any data structure with amortized time guarantees) work great if your batching lots of operations. Unfortunately if latency of individual operations matters, it's not the best solution.

The classic CS 101 algorithm of doubling an array to grow it has similar (though not as extreme) behavior. Most of the time additions are very fast. Every so often, you incur a large cost. There are ways around this, however on modern architectures block memory copies are fast enough that it's usually not a problem.

Norman Diamond
19 Jan 2006 9:44 PM
#
Thank you Mr. Vognsen and Mr. Lax. It looks like I don't need to spend time learning about splay trees ^_^ Hash tables, B-trees, etc., still seem to be enough.

hartwil
23 Jan 2006 11:50 PM
#

> Do you mean that anti-rebalancing is done each time an item is fetched?

Splay trees are not only well-balanced in many practical circumstances, but the most commonly accessed elements tend to be at the top. This is one of the big benefits over red-black trees. However, if you go through accessing all the elements, you mess up the structure that makes access of commonly used elements quick. And if you do it in order, as pointed out by Per Vognsen, you severely unbalance the tree.

Something that seems to be overlooked here is that splay trees are not necessarily the wrong structure to use in the story above. The problem is that the implementors failed to provide a function that walks the tree without splaying. That way, you could have a tree that performs better than a simple binary tree for single operations (caveat: amortized, and only in the most common case of unevenly accessed elements), and also better than a simple binary tree for walking the tree (average case, although if the tree is never abused by walking it with the splay operations, you could probably say always).

Splay trees may even be better for cache performance, because finding a commonly accessed element should rarely require descending through an uncommonly accessed element.

Perhaps the moral of the story is not that those developers tried to be too clever, but rather (quoted from an old fried) "you must be 10% smarter than an object to use it".