From: Bill Gates [/o=microsoft/ou=northamerica/cn=Recipients/cn=1648] on behalf of Bill Gates
Sent: Tuesday, May 06, 1997 4:10 PM
To: Nathan Myhrvold; Steven Sinofsky; Jon DeVaan
Cc: Aaron Contorer
Subject: FW: Applications boot time

I send this since the way I copied you on the other mail probably didn't include the enclosures where the technology is discussed.

Personally I think reducing the disk overhead of boot time by "logging accesses" after firing up applications is ridiculous. Office should understand how to organize the disk without the user having to go through this. Some shared DLLs we should make multiple copies of.

-----Original Message-----
From: Bill Veghte
Sent: Saturday, May 03, 1997 10:15 AM
To: Bill Gates; Marshall Brumer
Cc: Aaron Contorer; Carl Stork (Exchange); Mike Porter; Moshe Dunie; Jim Allchin (Exchange); Bill Krueger
Subject: RE: Applications boot time

We are pursuing aggressively for Memphis. The question we need to think about is what the coordinated strategy across MS you want us to pursue on the Intel work. We are combating a bloatware perception w/ Office and reducing Office load time would help diminish this perception. At the same time, the Office business is an incredible revenue producer so getting this added "benefit" could help. However, we need to factor in a couple of other elements as well tho as we think about this. They are:

a) The work that Intel has done certainly can benefit more then just Office. Appload time is an issue for the PC platform at large and by putting it in the OS this benefit is accrued broadly. If this capability is designed specifically around Office then the end-user will probably not benefit from the rest of their apps on their system. This need not be the case. The work that Intel has done is good enough so that the "optimization" does not degrade significantly across a set of applications (as opposed to a prioritized list). If Office implements the tool as a system wide benefit, call me a traditionalist but it just seems like an odd end-user experience. Install Office, then get prompted to walk thru a series of steps to identify the applications that you run (w/ Office apps at the top of the list of course) and then defrag your disk.
b) Attached below is a write-up from BillKru, who is leading our technical evaluation of Intel's work, on what Intel has done and how we would refine into a system offering. To achieve the best appload time optimization means a new app defragger which we are doing now, FAT(32), and changes to the way that Intel tool logs access to the disk. The Memphis team is committed to doing these refinements to these system tools in conjunction w/ the work from Intel. The end-user experience is along the lines of a system "tune-up" utility that sometime after upgrade recommends to the user a series of things to improve the performance & efficiency of their system. The most critical element is disk optimization. Subsequently, on a periodic basis, we will profile the users app usage over an extended period of time to ensure our optimizations are consistent with what the user is currently doing (eg. they may have installed an app).

intel exec report.doc

a) Intel's distribution capabilities: Intel of course is very anxious to get credit for their work. If they licensed w/ Office, we would have a harder time preventing them from licensing to everyone under the sun. This has two limitations; positioning Intel as a great sw company, mitigating the amount of "benefit" accrued directly to MS.

Plaintiff's Exhibit
7948
Comes V. Microsoft

-----Original Message-----
From: Marshall Brumer
Sent: Friday, May 02, 1997 2:17 PM
To: Bill Gates
Cc: Aaron Contorer; Bill Veghte; Carl Stork (Exchange); Mike Porter
Subject: RE: Applications boot time

We have completed an LOI to put the application launch technology for disks into Memphis. We have not completed an agreement that allows us to have the code here. The agreement is in process and should be complete soon.

The office folks can already start testing with the MS version of this code in Memphis. This will be the base that we use to apply the Intel technology to.

You need to decide if this becomes a feature for Memphis or something that the apps folks ship as part of their setup -

billv can add here.

-----Original Message-----
From: Bill Gates
Sent: Thursday, May 01, 1997 7:20 PM
To: Marshall Brumer
Cc: Aaron Contorer
Subject: FW: Applications boot time

Do we have the Intel code yet? Did we reach a good agreement on this?

Every day that goes by without this is bad. We need to get going.

-----Original Message-----
From: **Wael Bahaa-El-Din**
Sent: Thursday, May 01, 1997 12:42 PM
To: Bob Fitzgerald; Jon DeVaan
Cc: David Fields (NT); Stephen Hsiao; Bill Gates; Jim Allchin (Exchange); Moshe Dunie; David Cutler; Lou Perazzoli; Mark Lucovsky; Rick Rashid; Duane Campbell; Antoine Leblond; Mark Walker (Word); Jim Walsh
Subject: RE: Applications boot time

Jon and Fitz,

1. We can measure the gain with what Fitz has already (a list of hints to the loader to touch pages in a certain order). However, Fitz's method may read pages that doesn't get referenced (as he mentioned), this will not yield good performance for 16MB which is very sensitive to %waste. If we get a properly BBTized build for Word using the startup scenario, we can do the best possible experiment for the app set of images (minimize the pages read, read in big 32K or 64K). We can do the experiment as soon as the office group provides us with the proper BBTized image.

2. Startup is by far the most important operation impacting "perceived performance" We need to BBTize using startup (in addition to open/save/print) with improved weights in the scenarios and making sure that page placement is satisfactory by viewing the outcome with our tools. We would like to depend on your group and the BBT group to help us. We will initiate a meeting next week with our group, Jim Walsh, and Fitz.

3. Anyone thinking about reducing the number of pages that get referenced? **Yes!**
   a. We are BBTizing the NT5.0 kernel for the first time with a lot of help from Fitz and Hoi from the BBT group. This should reduce the code pages referenced dramatically in the paged sections of the Kernel.
   b. We are working on reducing the pages touched from the registry, page tables, non-paged pool, font files, system threads, etc.
   c. David Fields & NTW perf team will continue working with the Office people to give them feedback in optimizing for NT as they did with Mark Walker for Word 8.0 to reduce the boot from 13 to 6 seconds in 16MB.
4. In 16 Mbyte memory, we measure an average of 8KB/read (Fitz's number of 20 KB/read is measured in 32 MB memory where NT uses a larger cluster).

Thanks,

Wael

-----Original Message-----
From: **Bob Fitzgerald**
Sent: Wednesday, April 30, 1997 2:05 PM
To: Wael Bahaa-El-Din; Jon DeVaan
Cc: David Fields (NT); Stephen Hsiao; Bill Gates; Jim Allchin (Exchange); Moshe Dunie; David Cutler; Lou Perazzoli; Mark Lucovsky; Rick Rashid; Duane Campbell; Antoine Leblond; Mark Walker (Word); Jim Walsh
Subject: RE: Applications boot time

The reference scattering in winword (probably) means that code was getting executed during boot in the WinNT perf lab that wasn't getting executed at boot in the Office build lab. The existing lego training process (scenarios, platforms) may not adequately prepare the optimized binaries for the variety of target platforms they will ultimately run on. Some of this may be fixable.

We should consider the possibility that some fragmentation -- skew between training scenarios and real customers -- is unavoidable and that we need to tolerate some modest amount of fragmentation and still boot quickly.

I've done some approximations of gang loading -- reading all the necessary pages of winword.exe in address order, then all the pages of mso97.dll, etc. The key lessons are:

- large block reads (64 KB) maximize disk bandwidth. The ~5 page per read average with the WinNT 4.0 cluster pager (cluster size 7-8) delivers as little as 1/2 the disk bandwidth available with 16 page reads. (this depends a lot on the particular disk you measure)
- no overruns, as any unnecessary pages read increase memory pressure and increase paging. We currently overrun a lot.
- locality matters, little clumps of one or two pages drag down the average disk read size and bandwidth and increase overruns
- reading in address order instead of fault order increases the apparent locality, increases read size and decreases overruns

It may not be necessary to create a special "boot section" to make gang loading work -- just knowing the list of pages needed is enough. The list could live anywhere, e.g. in the registry as the Win97 folks may be considering. To prevent overruns, it is important to know when to stop, i.e. to keep the WinNT cluster pager from reading off the end of each island of contiguous page references. Clever disk layout may help to increase disk read sizes or to reduce seek delays.

Coalescing the dozen or so system .DLLs (kernel32.dll, user32.dll, gdi32.dll, shell32.dll, ole32.dll, ...) that are used into a single .DLL would decrease footprint and increase locality.

From:	Jon DeVaan
Sent:	Wednesday, April 30, 1997 10:25 AM
To:	Wael Bahaa-El-Din
Cc:	David Fields (NT); Stephen Hsiao; Bill Gates; Jim Allchin (Exchange); Moshe Dunie; David Cutler; Lou Perazzoli; Mark Lucovsky; Rick Rashid; Bob Fitzgerald; Duane Campbell; Antoine Leblond; Mark Walker (Word); Jim Walsh
Subject:	RE: Applications boot time

I'm surprised by the fragmentation of the accesses in winword.exe given that we already bbt word. My first guess is that compromises to boot are introduced as we add usage scenarios to the builds. It's an area definitely worth looking into.

Is anyone thinking about reducing the number of pages that get referenced? We did work in Office 97 to eliminate system calls made, but it looks like there ought to be many more to eliminate. I don't think we can do all of the work here though. MarkL had a list of redundant api calls in Office 95 that we made good progress in fixing in Office 97. That was very helpful. If you guys had a list of, "Why are you calling this api?" for apis that are suspicious to you, we could put it to good use.

I wonder if we can do a good simulation of the effect of boot block before we get the other things done. One experiment that ought to not be too hard to do, is to hack loadmodule to load the boot section. For the experiment just estimate a block using the existing exe. Word has a pretty reasonable set of 3 or 4 blocks at the beginning of the exe that could be called the boot sector. Also for the experiment, just use the module name to see if it is winword.exe, or mso97.dll and their associated intl dlls, then set the read ahead amount to the boot block size, touch the first page, set the read ahead size back, and touch all the other pages in the boot block. (Maybe there is a better way to make the read ahead pages not get discarded right away.) Seeing the effect of this would be very interesting. Of course my description of how to do this is nanve. If you see a better way, then great.

-----Original Message-----
From: **Wael Bahaa-El-Din**
Sent: Tuesday, April 29, 1997 8:13 PM
To: Bill Gates; Jim Allchin (Exchange); Moshe Dunie; Jon DeVaan; David Cutler; Lou Perazzoli; Mark Lucovsky; Rick Rashid; Bob Fitzgerald
Cc: David Fields (NT); Stephen Hsiao
Subject: FW: Applications boot time
Importance: High

This is a follow-up on BillG request to optimize app startup:

In the report enclosed below, we discuss both Word8 startup running on NT4.0 and how we can improve it. The startup takes 6.04 seconds on 100Mhz Pentium with 16 MBytes of memory because it involves reading 3 Mbytes from around 50 files on disk with a poor IO rate of 631 KB/s.We would like to be able to perform the startup by reading 2Mbytes (pages that are really referenced) with an IO rate of 2Mbytes/s. The startup also involves CPU consumption of 1.5 seconds. We show that the result of the current BBTizing (due to the scenarios used) can be significantly improved. With the improved BBTizing and a change to use larger clusters during application startup, we should be able to get a much higher bandwidth for disk reads (around 1.5-2MB/s) for an overall boot time of 3.0 seconds in 16 Mbytes. After working on the proper BBTizing of WORD image (and other office applications), we will investigate other solutions to improve the I/O bandwidth such as reorganizing the on-disk locality of the files and constructing a boot section for the application code/data.

Thanks,

Wael

P.S. Last figure shows the current impact of BBT on WORD.exe. It doesn't print on some printers but you can see you on your monitor.

<< File: word8bootupdate.doc >>

-----Original Message-----
From: **Moshe Dunie**
Sent: Tuesday, March 18, 1997 9:25 AM
To: Lou Perazzoli; Wael Bahaa-El-Din
Subject: FW: Applications boot time
Importance: High

Can you please do this experiment with the office folks.

Thanks.............Moshe

-----Original Message-----
From: **Bill Gates**
Sent: Tuesday, March 18, 1997 8:54 AM
To: Moshe Dunie
Cc: Jon DeVaan; Jim Allchin (Exchange)
Subject: FW: Applications boot time

Can you have someone from NT work with the Office group on point #1 here?

I think Office boot times are critical to our future and I am pushing for more innovation in this area.

-----Original Message-----
From: **Jon DeVaan**
Sent: Monday, March 17, 1997 10:13 PM
To: Bill Gates
Cc: Richard Fade; Steven Sinofsky; Brad Silverberg; Nathan Myhrvold; Aaron Contorer; Rick Rashid
Subject: RE: Applications boot time

Two things would be extremely helpful for making this come true.

1) I am embarrassed to report that we still do not have agreement from the OS teams to declare a boot section in an exe and load it all at once. This would be a major improvement. (OK, perhaps a wild assertion on my part) The argument against this is usually along the lines of "we tried writing a tight loop that paged in x bytes of code in the app and it didn't help boot time any." My argument against this is, that experiment does not cause x-bytes to happen with exactly one IO operation. I want the NT guys to run this experiment: Change the gang-load size parameter to be the boot size of an exe for the first fault, then change it back dynamically. This is the right experiment to run. I can't convince anyone to do this experiment.
2) We need Lego v. 2. Lego has been a big help, but it has a bunch of inadequacies. I was surprised to learn that it cannot do code groupings based on scenario. What I mean is, I want to know for n operations (boot, file open, file save, file print) the set of basic blocks used in each operation. Then I want the code in my exe distributed so that the code that is boot only is one contiguous block, boot AND file open in one contiguous block next to that, the code that is boot AND file save next to that, etc..., all n! blocks defined. Then I want those blocks ordered so that by priority of operation I have one contiguous block of code for the highest priority operations and then 2, 3, 4, or more blocks for operations as priority wanes. Lego can't do this today.

It is also fair to note 2nd boot of an app on win95 or NT are typically 4-5x faster than first boot. (i.e. 80% of boot time is page faulting)

-----Original Message-----

From: **Bill Gates**
Sent: Sunday, March 16, 1997 10:20 AM
To: Jon DeVaan
Cc: Richard Fade; Steven Sinofsky; Brad Silverberg; Nathan Myhrvold; Aaron Contorer
Subject: FW: Applications boot time

One goal I think has to be totally crucial for Office 9X is to get boot times well below 10 seconds.

I know this will require invention and work with the OS and even rethinking how we use DLLs but I think it's a requirement.

Office feels heavy for a number of reasons but the one that you really notice is the applications boot time.

-----Original Message-----
From: **Rick Rashid**
Sent: Saturday, March 15, 1997 12:48 PM
To: Bill Gates; Aaron Contorer; Darryl Rubin
Cc: Jim Allchin (Exchange); Steven Sinofsky; Butler Lampson; Nathan Myhrvold
Subject: RE: Applications boot time

I'll look into this again, but it was my impression that with the last round of LEGO work which already allows Office to linearize its initial page faults and with way NT handles paging that we were already getting about all we could get in terms of loading speed. Assuming fairly linear accesses, a disk should be just as good as a 100MB ethernet and probably better.

The biggest loading issue, I suspect, is related to the fact that the "access set" of a Windows NT system with Office is much larger than 24MB (actually its larger I believe that 32MB) and that there is going to be paging going on other than just paging in the application.

Also, I believe, there is considerable CPU time (seconds) devoted to "startup" in the apps as they open files, review registry entries, link things, allocate space, etc. Nothing done to data load times will help make this go away.

I've also noticed that there is also a lot of "hidden" access to servers and devices which typically justs times out. When I run Office on the machine I have which has a zip drive, for example, it routinely spins up the drive for no obvious reason. Likewise I will often hear a random floppy access or see the system pause when I'm not connected to a network.

From:	Darryl Rubin
Sent:	Friday, March 14, 1997 10:18 PM
To:	Bill Gates; Aaron Contorer
Cc:	Jim Allchin (Exchange); Steven Sinofsky; Butler Lampson; Nathan Myhrvold; Rick Rashid
Subject:	RE: Applications boot time

There are also ways to improve the illusion of startup speed. It should be very easy for an app to put up what looks like the app window and the first page of the document (or the page the user last visited), even if this is mostly a smoke-and-mirrors show until more of the app loads to make it live. The app could also be restructured to prioritize which parts of the UI come alive first, based on what operations are the first that users usually try (scroll? Pull down file or edit menu?). Of course we should be making the initial working set of the app smaller and also do caching tricks like you suggest. I think that plus tricks could result in a dramatic improvement in perceived boot performance.

-----Original Message-----
From: **Bill Gates**
Sent: Friday, March 14, 1997 9:35 PM
To: Aaron Contorer
Cc: Jim Allchin (Exchange); Steven Sinofsky; Darryl Rubin; Butler Lampson; Nathan Myhrvold; Rick Rashid
Subject: Applications boot time

I am hard core about trying to find ways to make our applications boot faster. We have to do it. It's the whole reason people think our applications are too big.

The question I have is what if the server had say the most commonly used 24 megabytes of Office in Ram in a form that made it very easy to get to. Would it be faster over a 100megabit fairly unloaded Ethernet to get these bits across the network? The idea is basically the Berkeley NOW approach except without the low latency network which makes it such a big win for them. I wonder what tricks might allow this to work well. Reducing latency is a worthy project for many reasons.

Summary of MS Intel Meeting Re: Application Load Time Minimization

David Alles, Tony Ka, and Bill Krueger met with Intel earlier this week for a technical discussion of Intel's application load time optimizer implementation.

Summary of Intel's Disk Optimization Utility

Intel's app load optimizer is a standalone utility. It isn't integrated into any existing disk tools or the OS as a whole. Intel's approach is taken out of necessity since they don't have access to the sources for MS disk tools, MS core system components, or any disk tools in general. Running MS Defrag, or any third party disk defragger, without subsequently rerunning Intel's optimizer app will undo the optimizations. This isn't true of the MS optimizer since it's an integral part of Defrag to use the optimization log. I'll describe the major differences and phases of operation of the utility and what goes on in each.

MS & Intel Optimizer Technology Summary

The MS optimizer logs the access pattern of each app during the time it is loaded. It then organizes each cluster in the app in the sequence that it is accessed during load time and places these clusters so that they are physically contiguous on disk Additionally, the MS optimizer "ranks" the importance of an app when doing the optimization. Each app, after the first app in a set of apps to be optimized, has progressively less load time reduction, depending on its rank order, than it would were it optimized alone. Given a large number of apps, the final app sees little benefit of the optimization. This arises primarily due to accesses to shared DLLs. Using this placement strategy for a single app, there is really only one "best" way to organize a DLL that the app uses.

Intel's algorithm uses a mathematically sophisticated technique (Markov transition graphs) to organize the load time clusters of a set of apps so that the cumulative access time for the all apps is minimized. While the sacrifices some load time performance for a single app, all load times for all the apps in the set is reduced. We have not obtained a copy of Intel's optimizer to verify this behavior in a system we consider "typical". Intel's statistics indicate that they are able to optimize for ten apps that that the load times for the first app in the set did not vary significantly. Intel's approach is clearly better for a suite of applications like Office where it is desirable to improve the load time for all applications in the suite. However, we really need to quantify this behavior because Intel's test systems and 233 MHz Pentium MMX systems with 32 Mbytes of memory.

Technical Aside: Markov Transition Graphs

What Intel's optimization technique does is to analyze the disk access pattern of all applications in a set of apps being optimized and build a graph that represents the overall access pattern. Each node in the graph maps to a cluster. Each of the graphs vertices maps to a "weight" that indicates the probability of a transition from one node in the graph to the next. Intel refers to this type of graph as a Markov transition graph. The graph is used to perform two levels of optimization. The first level of optimization is used to locate cluster strings (i.e. cluster chains that are accessed in sequence) so that they are organized for optimal locality of reference. The next level of optimization is to locate the cluster chains with respect to one another so that they are optimally located with respect to temporality of reference. Intel's optimization algorithm is both mathematically sophisticated and computationally intensive. It takes 30-60 seconds of compute time on a 233 MHz Pentium using MMX instructions and a lot of memory (on the order of megabytes). The net result of all this analysis is that the reorganized clusters absolutely minimize the seeks between clusters for all the apps in the optimized set. It is seek time that is the dominant factor in disk throughput. Minimizing seeks minimizes access times.

Application Load Time Optimizer Startup

Superficially, at the user interaction level, Intel's and MS' optimizer are not all that functionally different from one another. Both collect information from the user about which apps the user wishes to optimize. Both start up the apps automatically, collect log information, and automatically shut them down. Intel's optimizer does not require that the user rank the apps to be optimized in order of importance as Microsoft's does. This isn't necessary given the algorithm Intel uses to optimize file cluster placement.

Based on usability tests we've recently done, not asking for the apps rank is desirable because it's one less point of potential confusion for the user. If we adopt intel's technology and do not need rank order, it may be possible to do away with asking for which apps to optimize altogether. Work is being done on a shell hook that records which apps are run and how frequently they are run. The MS wizard could just use this information and do the optimization based on usage data that's automatically collected. We may still want to let the technically adept pick and choose, but for the majority, full automatic optimization would be highly desirable.

Post User Input

The next phase in Intel's process it to completely defragment the disk. They do this to establish a baseline order for the disk. They spawn MS Defrag and it runs invisibly within their app. Once the disk is defragmented, they launch each of the selected apps minimized in order to collect the disk access patterns. Launching the apps minimized is also a nice touch, it is less confusing for the user because there's less going on visually. However, if an app hangs for some reason it may not be apparent to the user what the problem is. Intel has also encountered the same problems that we have experienced with shutting down applications: some don't shut down cleanly. There is no perfect general solution to this problem.

There is a compromise that Intel made here because it does not have the source for a disk defragger. After the disk is defragged, and those portions of the applications that have been optimized have been moved, the portions of the app that were not accessed during the load process are "stranded". By this I mean that the portions of the app unused at load time are not moved in close physical proximity to the optimized portions. For those sections of the app, the fragmentation has become worse. Running those app features not needed at load time will take longer to load because additional seeks will have to be done to reach those portions of the executable. By integrating Intel's technology into the defrag process, MS can ensure that the portions of the executable that are unused at load time are still placed in as close a physical proximity as possible to the load time working set.

Disk Access Logging

There is a significant difference in how MS and Intel collect the disk access information from the applications being run. MS' optimizer logs disk accesses by hooking at the IFS manager level and monitoring all logical file accesses. Intel's app hooks below IOS and logs sector level accesses. Logging only sector level accesses, as Intel's app does, makes the job of selecting which sectors to move significantly more difficult and does not provide all the necessary information to decide whether to move a specific cluster or not.

Why is it more difficult to use the sector level log?

The difficulty arises out of the fact that the Intel app, or MS Defrag for that matter, cannot move three classes of files. Determining whether a file belongs to the first two classes is merely a lot of work but do-able using a sector log. Determining whether a file is a member of the third class is not possible with a sector level log.

The first two classes of immovable files are the swapfile and files with SYSTEM and HIDDEN attributes. The penalty for accidentally moving parts of these two classes of files varies. Moving parts of the swapfile would appear to the user to cause GP faults or system hangs. Moving a SYSTEM and HIDDEN file may cause an app to refuse to run. This latter annoyance is due to copy protection schemes.

Technical Aside: Associating a Sector With A File

Here's a synopsis of the work required to determine if a sector belongs sector is movable using a sector log. The Intel optimizer has to grovel the disk in order to determine whether the sector belongs to the FAT, a directory, or to a file. FAT sectors aren't moveable. Moving directory sectors is undesirable because doing so makes name based file APIs perform more poorly and they are already at a disadvantage performance wise. A sector belonging to a file MAY be moveable. To determine whether a sector belongs to a file, it is converted into a cluster. The cluster chain for the file must be traversed to determine what the file's first cluster is. Knowing the first cluster of the file, one then has to search the directory structure of the disk in order to locate the file's directory entry in order to determine what the file's attributes are. To determine whether the file is the swapfile, one has to check the file name.

The third class of immovable file is a writable memory mapped file. There is absolutely no way to determine from a sector level log whether a file is memory mapped. Only the IFS and memory manager are privy to this specific information. The penalty for accidentally moving a cluster of a writable memory mapped file is potential file corruption. Unfortunately, Intel's app has this problem with this class of file and they are not aware the problem exists. Intel tests their app in a very controlled environment: only the disk optimizer app is running. This isn't a particularly realistic test environment and is indicative of why Intel claims that the app seems to be very reliable. We'll point out this problem to them so that they can address it in their optimizer.

The one other subtle problem associated with sector logging: all access that are satisfied out of the file system cache are missed by the sector logger because no sector reads are ever sent down to IOS in this case. While this omission isn't fatal, it does affect the frequency-of-access data that the logger collects for the Markov graph and that the optimizer uses. A better optimization might result of all these missed accesses were fed to the optimizer.

In any event, a logical log has none of the above sector-association work associated with it, and all of the detailed open-mode and memory mapping information is available to logical access logger via data the IFS manager makes available to the logger. That, along with the fact that a logical log is portable across systems, is why we chose this technique. We define *portable* as the ability to take a log generated for a set of apps installed on one system, and use that identical log to optimize another system which may have a totally different disk geometry.

Changes needed to MS App load optimizer, Defrag, IFS, and VFAT

Current MS App Load Optimizer Architecture

The MS app load time optimizer is implemented as three components. The following description gives an idea of the current implementation so that it may be contrasted with the changes that we intend to make.

1. The first component is the "app load time optimization wizard". The wizard walks the user through the process of selecting which apps to optimize. It then loads a vxd which will log subsequent files accesses. The wizard enables the logging vxd and then automatically launches and shutsdown the selected apps. Once the last app has been launched and shutdown, the

wizard causes the vxd to unload, thus ending the logging process. As the final step, the wizard launches Defrag.

2. The second component is an "hook vxd". This vxd is dynaloaded by the wizard. Once enabled, this vxd logs all read/write accesses made to any file. The log file is closed at the time that the IFS hooking vxd is told to unload by the wizard.

3. The third component is "MS Defrag". Defrag will use the log file, if one is present, to optimize the disk. If Defrag is started before the wizard, it will launch the wizard. Hence, it is not possible to accidentally defrag the disk and lose all your app load optimizations. The modifications made to Defrag were modest. The modifications entailed adding a preprocessor. The preprocessor reads the log file, eliminates immovable files, creates a list of cluster placement rules for defrag, and passes these rules to the existing defrag engine. Defrag then organizes the disk according to these rules.

New MS App Load Optimizer Architecture

The intent of this reorganization is to provide the capability to replace the portion of the above architecture that creates the cluster placement rules from the file access log. With a few simple architectural modifications, we will implement the cluster placement logic which is now a "hard-coded" part of Defrag as a DLL. The new architecture will consist of the same three components with the distribution of work slightly altered.

1. The first component will still be the "app load time optimization wizard". The wizard walks the user through the process of selecting which apps to optimize. It then loads a vxd which will log subsequent files accesses as it causes the selected apps to be loaded. The wizard launches the apps and shuts them down automatically. Once the last app has been launched, the wizard causes the vxd to unload, thus ending the logging process.

   At this point the wizard will open the log file. It will apply the "immovable file filtering rules" to entries in the log file eliminating references to those which cannot or should not be moved. It will create a set of data structures that describes the file access patterns in a form that is directly usable by the Intel optimizer. The wizard then calls the single service provided by the DLL. This service is nothing more than the logic that Intel has implemented to create and manipulate the Markov graph. The service will return a set of data structures that will indicate how the clusters associated with the files in the log are to be organized relative to one another. The wizard will write this data to an intermediate file that will be directly usable by defrag. *The reason that we do this here is that Intel's optimizer implementation is 32-bit code and so is the wizard. Unfortunately, Defrag is 16-bit code. Structuring things this way simplifies life a great deal.*

   As the final step, the wizard launches Defrag.

3. The second component is still an "IFS hook vxd". This vxd is dynaloaded by the wizard. Once enabled, this vxd logs all read/write accesses made to any file. The log file is closed at the time that the IFS hooking vxd is told to unload by the wizad.

The only difference is that the logger records additional information that the IFS manager makes available to it. The additional information is needed by the Intel optimizer and frees the Intel optimizer from ever having to grovel the file system directly and understand file system geometry and data structures.

The third component is "MS Defrag". Defrag works just like before. Only now is has even less work to do since none of the code which created the optimization lists is a part of defrag any longer: it was moved to the wizard. Defrag just opens the intermediate file and begins its work using the placement rules contained in this file.

This modified architecture allows us, Intel, or any third party for that matter, to replace the cluster placement logic without touching a line of code in the wizard, logging vxd, or defrag. From the standpoint of ease of enhancement and robustness, this is a *very* worthwhile abstraction for the following reasons.

a. All of the trickiness of hooking the IFS/FSD/IOS correctly and efficiently is done by the hook vxd.
b. All of the rule-correctness for eliminating immovable files is done in the wizard.
c. All of the complexity of determining when to restart the defrag process if an app modifies the disk is done by defrag.
d. All of the finesse in moving clusters around the disk in a way that won't cause file corruption if power is lost or an app GP faults is already done by defrag.

The value that Intel's process adds is done in choosing the optimal placement of clusters, relative to one another, for the set of chosen apps. In fact Intel is very interested in continuing refinements on their optimizer: there may be additional wins that they can make in reducing load time based on information that they have about the performance of various classes of disk drives. Intel can replace their DLL, trying improvements in the placement logic, without needing any new components from MS.

Disclosing these formats and protocols to third party disk tool vendors focuses their efforts in adding value by coming up with superior placement algorithms rather than twiddling disk bits.

MS & Intel Organization of Work

We will have to do some restructuring of our app load logging component, the defrag engine, and enhance some of the data that the IFS and VFAT return to the app logger. The changes are not drastic and will result in some additional flexibility in how we can upgrade or replace the optimization back end of Defrag as I've already described.

BillKru will modify the IFS and VFAT to return some additional information about file accesses to the app load time wizard. The extra information will simplify the decision making process in determining whether a file is moveable for MS app load wizard's, MS defrags, and Intel's optimizer DLL. Fundamentally, we want to make it completely unnecessary for the Intel optimizer to have to have any detailed information about internal file system formats which it now has to have to do its work.

Tony will modify the App logging Wizard to utilize the additional file access information supplied by the IFS manager and VFAT. He will also modify the wizard to: containing the filtering logic which is now a part of defrag (b) use a DLL interface to process the information collected by the app logging and (c) write the information that the Intel DLL returns out to an intermediate file for defrag immediate use.

Aaron will modify defrag. This work entails nothing more that stripping out code that was added and reading and using the intermediate file rather than the text version of the log file. Interestingly enough, even this version of the placement file is portable across systems.

Intel will have to extract their cluster placement optimization code into a DLL. Some modification of data formats will be necessary and we've already begun this work with them.

What MS Wants From Intel

We are interested in only that portion of Intel's work which creates the optimal relative organization of working set clusters for sets of applications. There is no other portion of Intel's application that provides any technology that we don't already have, or is better integrated into MS disk tools.

MS wants the source code to be able to perform this function and will implement it as a DLL which contains a callable "optimizer" function before Memphis ships. MS will use this DLL architecture in the final retail release version of Memphis. There is no technical reason to integrate the optimizer code in the DLL within Defrag itself. Maintaining the optimizer as a separate DLL for this specific function has some technical and logistic benefits both during development and afterwards as has already been described. You might think of the DLL as an optimization "back end" that could be replaced with an enhanced version at a future time without replacing the front end, in the case the app logging vxd and the Defrag engine. Prior to shipping Memphis MS will require source code for the DLL so that it can be included in our build process, built for release with the retail version of Memphis, and bugs in it corrected on a timely basis.

Intel will convert their current optimizer logic into a DLL that will be callable by the MS wizard. The optimizer DLL will need to be modified to run on both MMX and non-MMX CPUs. Currently, the Intel technology uses MMX instructions. Intel should do these modifications since they are the most familiar with their code base.

During the period where MS and Intel are actively developing and integrating our two respective technologies, Intel will supply MS with the optimizer DLL which does only this one function. MS will modify its wizard, applications logger, and log preprocessor. They will supply this DLL with an array of data structures that contain access pattern information for those clusters that are to be optimized and information which Intel's optimizer needs regarding those clusters within this pattern of accesses that are "immovable". The Intel DLL will process this information and will return an array of data structures that will indicate the optimized placement of these clusters relative to one another. The MS Defrag engine will use this relative cluster placement information to place the clusters to be optimized in an area of the disk that Defrag either finds available, or creates.

We are working with Intel to define the input data structures that are required by their optimization function and the output data structures that are required by our Defrag engine. We're also working on a joint schedule for development and integration of our respective components. Both a preliminary schedule and data structure and interface definition were sent to Intel on Friday April 25.

What Intel Wants From MS

Attribution for the portion of the work that they've done in the form of sticking their logo in some screen during the optimization process. Probably some joint press work.

Intel will need timely drops of the three components that MS supplies and are described above.