[Moving A Database to New Storage With No Downtime](http://www.bobpusateri.com/archive/2013/03/moving-a-database-to-new-storage-with-no-downtime/" \o "Moving A Database to New Storage With No Downtime)

**The Problem**

At work we have been very fortunate to get a new SAN, and I was charged with moving data from the old SAN to the new one. This data falls under 2 main groups: large numbers of small files which can be moved at any time without consequence, and data files from the production SQL Server database. I covered how I moved the non-database files in a [previous post on the Microsoft FCIV](http://www.bobpusateri.com/archive/2013/02/verifying-file-copymove-operations-with-microsoft-file-checksum-integrity-verifier/). This time I’ll cover the larger task of moving the databases. To make things much more interesting, I did not have the luxury of being able to take any downtime to accomplish this move. Making use of pre-scheduled maintenance windows was fine, but those are typically 2 hours every 2-3 months, not nearly enough time to move the 22TB of data that was in need of migration.

Just to be clear, the instance of SQL Server was not moving or changing at all. The server was connected to both the old and new SANs, and the objective was to move a particular database from old to new. Yes, there are other databases on the server, but they are located on other storage and did not need to be moved. We use SQL Server 2008 R2 Enterprise Edition, so at least I had the full arsenal of functionality available to me.

I began listing out all the ways to move data that I could think of, and started disqualifying from there:

**Copy the files**. This would be easy and relatively fast, but requires taking the database offline to move the files. Since uptime is paramount, this is a no-go.

**Online index rebuilds**. Using enterprise edition means it’s possible to rebuild tables and indexes online without interrupting users. It is also possible to move objects to a new filegroup online as well. This is very useful, but comes with some limitations. Prior to SQL Server 2012, indexes containing LOB (large object) information such as columns of type image, ntext, text, varchar(max), nvarchar(max), varbinary(max) and xml can’t be rebuilt online. SQL Server 2012 has improved this feature so this limitation only applies to image, ntext, and text columns, which are deprecated anyway. But even though SQL Server 2012 lets you perform online rebuilds of tables with LOB data, it still won’t let you move the LOB data to a different filegroup. This solution gives us the uptime we require, but unfortunately can’t move everything.

**Mirroring**. Much like a [rolling upgrade](http://msdn.microsoft.com/en-us/library/bb677181.aspx), mirroring can be used to migrate a database between instances. You can set up a mirroring session with the mirror server on the instance you want to migrate to, manually fail over, and you’re now on the new instance.  Unfortunately it won’t work in this case because the primary and mirror databases can’t reside on the same instance.

**Backup and restore onto the new SAN**. This is a very strong candidate, especially since I can backup the database in about 3 hours and restore it almost as quickly. Restoring a backup on the new storage (with a different database name) can take place while the current database remains online. Cutting over would involve backing up the tail of the transaction log on the old database, restoring it on the new one, and then swapping names. The cutover would be quick enough to fit in a maintenance window, making this a very attractive option. I ended up choosing against it because the new storage would be partitioned differently than the old storage  and adjusting the files and filegroups and then rebuilding objects across them would make the process more complicated than the solution I ended up choosing.

**Add and remove files from the filegroup**. This approach can be very useful, but rarely gets mentioned. SQL Server allows you to add and remove files from filegroups at almost any time, and this can be leveraged to move data between different storage systems. What makes it awesome is that the operation is completely online with no interruptions to the user, all types of data (including LOB data) is moved, *and* it works in any edition of SQL Server. As always, there are trade-offs. Shrinking data files is generally frowned upon as it causes [high levels of fragmentation](http://www.sqlskills.com/blogs/paul/why-you-should-not-shrink-your-data-files/). It’s also very slow, and [LOB data makes it even slower](http://www.sqlskills.com/blogs/paul/why-lob-data-makes-shrink-run-slooooowly-t-sql-tuesday-006/). Still, fragmentation can be solved using REBUILD or REORGANIZE, and if speed is not important, it could be the way to go.

**My Solution**

The add/remove file method is best for preserving uptime, but due to its slow speed I decided to use a hybrid solution with 4 parts:

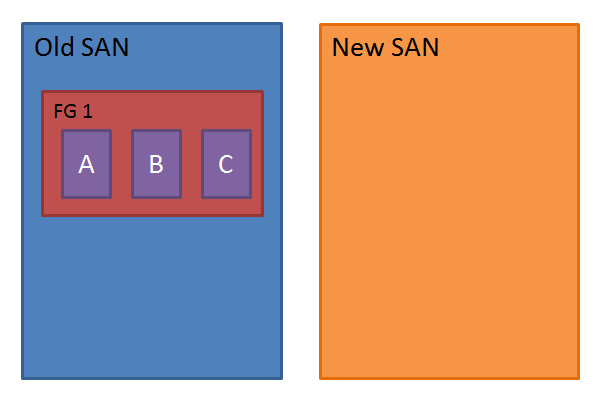
**1.** All tables & indexes that can be rebuilt online on a new filegroup are moved to a temporary filegroup on the new storage.

**2.** The remaining tables & indexes are moved via adding & removing files to the original filegroup.

**3.** The contents of the temporary filegroup are moved back into the original filegroup via online rebuilds.

**4.** The transaction log and primary filegroup (which cannot be moved by the add/remove method) are moved during a scheduled downtime.

The image below shows our starting state: the new SAN is empty, while the old SAN has a (non-primary) filegroup [FG1] with files A, B, and C in it, each 1GB in size. Most systems will contain more filegroups, but for simplicity I only show one here.



The first step was **online index rebuilds**. For each filegroup on the old SAN I created a temporary one on the new SAN. For example, in this case of having [FG1] on the old storage, I’d create filegroup [FG1\_Temp] on the new storage. Then I did online rebuilds onto the temp filegroup for each table and index that would allow it. This can be an arduous task for filegroups with large numbers of objects, so I created a powershell script to automate this which I’m happy to share. Creating the new filegroup and files can be done via the following T-SQL:

[?](http://www.bobpusateri.com/archive/2013/03/moving-a-database-to-new-storage-with-no-downtime/)

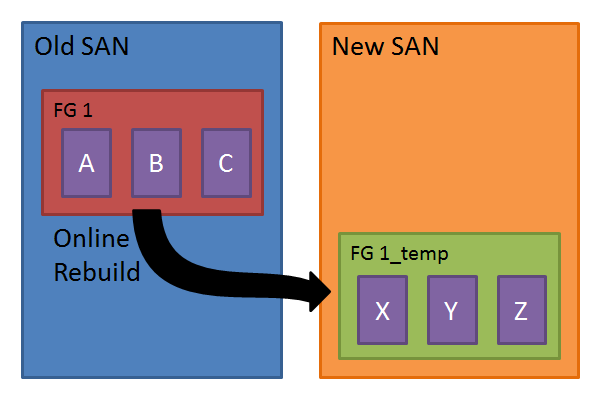
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | ALTER DATABASE DB\_Name ADD FILEGROUP FG1\_Temp;    ALTER DATABASE DB\_Name ADD FILE  (NAME = 'X', FILENAME = 'path\_to\_san2\X.ndf', SIZE = 1GB, FILEGROWTH = 1GB),  (NAME = 'Y', FILENAME = 'path\_to\_san2\Y.ndf', SIZE = 1GB, FILEGROWTH = 1GB),  (NAME = 'Z', FILENAME = 'path\_to\_san2\Z.ndf', SIZE = 1GB, FILEGROWTH = 1GB)  TO FILEGROUP FG1\_Temp; |

The powershell script will move objects online from one filegroup to another. Objects which cannot be moved online will be ignored. When you open the script file you will see variables that need to be filled in for the server name, database name, and source and destination filegroups, and a folder to log to. Optionally you can filter by schema name, object name, and size.

I am happy to share this with the community, but please be advised of the following: This script needs to run in[sqlps](http://msdn.microsoft.com/en-us/library/cc280450.aspx), the SQL Server powershell extension. It was written for and tested on SQL Server 2008 R2. I have not tested it on SQL Server 2012. I do not consider this to be perfect or in any sort of a finished state. **And as always it is your responsibility to read and understand this code before running it on your own systems. Always test in a development environment, no warranty is expressed or implied, etc. etc., use at your own risk!**

[**You can download the script here**](http://img.bobpusateri.com/bc/2013/03/OnlineFilegroupMigration.zip)**.**

After the online filegroup migration, the storage layout will appear as follows, with a temporary filegroup on the New SAN:



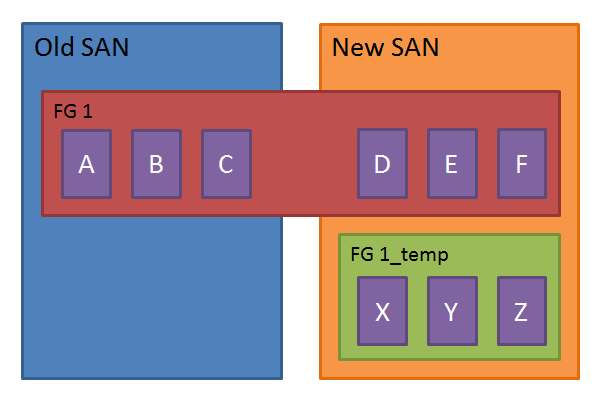
Once the migration script has completed, the old storage will contain the original filegroup with all the objects which could not be moved online. At this point all those objects still need to be moved onto the new storage, and the best (and perhaps the only) way to do this in an online fashion is by**adding and removing files from the filegroup**.

Here’s how it works: SQL Server allows the addition and removal of files from filegroups at almost any time via the [ALTER DATABASE](http://msdn.microsoft.com/en-us/library/bb522469.aspx) command. (You can’t add or remove files while a backup is running.) A filegroup can be extended onto new storage by adding files to it which are located on the new storage. The script below shows how to do this by adding files D, E, and F, located on SAN 2.

[?](http://www.bobpusateri.com/archive/2013/03/moving-a-database-to-new-storage-with-no-downtime/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | ALTER DATABASE DB\_Name ADD FILE  (NAME = 'D', FILENAME = 'path\_to\_san2\D.ndf', SIZE = 1GB, FILEGROWTH = 1GB),  (NAME = 'E', FILENAME = 'path\_to\_san2\E.ndf', SIZE = 1GB, FILEGROWTH = 1GB),  (NAME = 'F', FILENAME = 'path\_to\_san2\F.ndf', SIZE = 1GB, FILEGROWTH = 1GB)  TO FILEGROUP FG1; |

After executing that, filegroup [FG1] is now spread across the old and new storage with 3 files on each SAN, as well as the 3 files containing the temporary filegroup [FG1\_temp] on the new SAN:

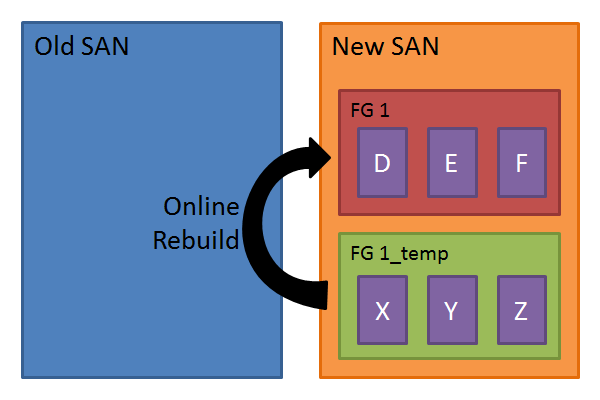


At this point, files D, E, and F contain very little data, if any at all. Merely creating a file doesn’t mean it will be populated with existing data, though SQL Server will start to make use of it for operations that allocate space on disk from here on out. The objective now is to remove files A, B, and C, so the filegroup has no presence on the Old SAN. This is done using the [DBCC SHRINKFILE](http://msdn.microsoft.com/en-us/library/ms189493.aspx) command with the EMPTYFILE argument, which will remove all of the data from the file by distributing it across the other files in the filegroup. Depending on the size of the file and the amount of data it contains, this operation can take a *long* time. Fortunately if you have to stop it, the work that has already been performed will not be lost. Once a file has been emptied in this manner it can be removed by using ALTER DATABASE. The code to accomplish this is as follows:

[?](http://www.bobpusateri.com/archive/2013/03/moving-a-database-to-new-storage-with-no-downtime/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | DBCC SHRINKFILE (A, EMPTYFILE);  ALTER DATABASE DB\_NAME REMOVE FILE A;    DBCC SHRINKFILE (B, EMPTYFILE);  ALTER DATABASE DB\_NAME REMOVE FILE B;    DBCC SHRINKFILE (C, EMPTYFILE);  ALTER DATABASE DB\_NAME REMOVE FILE C; |

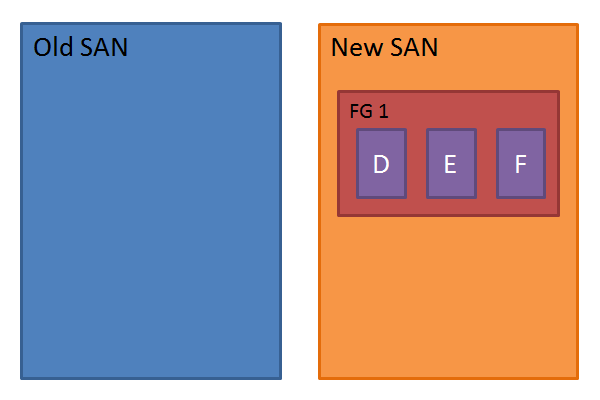
When this is complete, filegroup [FG1] will only exist on the new SAN, however there is now 2 filegroups and we started with 1. Filegroup [FG1\_Temp] can be removed once all objects in it have been moved back into filegroup [FG1], again using the online migration script. I realize this method involves rebuilding some objects more than once, however in my testing it was still faster than moving everything by adding/removing files from the filegroup. I would imagine your mileage will vary depending on your environment and the number of tables containing LOB data, but this is what worked best for me. On my system about 80% of objects could be moved online, leaving only 20% that needed to be moved via the slower method.



Once the online rebuild script completes, things will be nearly complete. All data will be in its original filegroup on the new SAN. Filegroup [FG1\_Temp] and its files can now be removed via the following script:

[?](http://www.bobpusateri.com/archive/2013/03/moving-a-database-to-new-storage-with-no-downtime/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | ALTER DATABASE DB\_Name REMOVE FILE X;  ALTER DATABASE DB\_Name REMOVE FILE Y;  ALTER DATABASE DB\_Name REMOVE FILE Z;    ALTER DATABASE DB\_Name REMOVE FILEGROUP FG1\_Temp; |



Once this is repeated for each filegroup in the database, the job will be almost complete. The primary filegroup and transaction log cannot be moved using either method just discussed, so a downtime will be required to move them to the new storage. I was able to make use of existing maintenance windows to detach the database, copy the files to the new SAN,and reattach it. Everything came back up without incident.

At this point the only issue left to tackle is fragmentation. The data that was moved via online rebuilds is not affected, but tables and indexes moved via the add/remove files method will be very highly fragmented. The quickest way to resolve high levels of fragmentation is to rebuild an index, but these are the indexes which cannot be rebuilt online! If an offline rebuild is not possible (and since uptime was very important in my case it wasn’t possible) the next best method is to use the REORGANIZE argument of the [ALTER INDEX](http://technet.microsoft.com/en-us/library/ms188388.aspx) statement. REORGANIZE is typically slower than REBUILD, but it will allow the index to remain online for users while it runs.

**Final Thoughts**

I don’t consider this method to be perfect by any means, but it worked rather well for me given the uptime requirements I had to meet. Thank you very much Jen for the excellent topic, and thank you to all of you who kept reading to the end! I hope this proves helpful!