Migrating a (Large) Science Database to the Cloud

Ani Thakar
The Johns Hopkins University
Institute for Data Intensive Engineering and Science,
3701 San Martin Drive,
Baltimore MD 21218-2695
(410) 516-4850
thakar@jhu.edu

Alex Szalay
The Johns Hopkins University
Institute for Data Intensive Engineering and Science,
3701 San Martin Drive,
Baltimore MD 21218-2695
(410) 516-7217
szalay@jhu.edu

ABSTRACT

We report on attempts to put an existing scientific (astronomical) database – the Sloan Digital Sky Survey (SDSS) science archive [1] - in the cloud. Based on our experience, it is either very frustrating or impossible at this time to migrate an existing, complex SQL Server database into current cloud service offerings such as Amazon (EC2) and Microsoft (SQL Azure). Certainly it is impossible to migrate a large database in excess of a TB, but even with (much) smaller databases, the limitations of cloud services make it very difficult to migrate the data to the cloud without making changes to the schema and settings (for example, inability to migrate a spatial indexing library, and several other user-defined functions and stored procedures) that would invalidate performance comparisons between cloud and on-premise versions.

Without being able to propagate the performance tuning and other optimizations to the cloud, it is perhaps not surprising that the database performs poorly in the cloud compared with our on-premise servers, but preliminary performance comparisons show a very large (an order of magnitude) performance discrepancy with the Amazon cloud version of the SDSS database. We have also not yet investigated the performance tweaks that could be possible within the cloud.

Although we managed to successfully migrate (a subset of) the SDSS catalog database to Amazon EC2, once it was in the cloud we were not able to access the database in a meaningful way from the outside world. Even though this was advertised as a public dataset on the AWS blog, it was not clear how other users or the public would be able to access this data in a meaningful way, if at all.

These difficulties suggest that much work and coordination needs to occur between cloud service providers and their potential database clients before science databases can successfully and effectively be deployed in the cloud. It is important to note that this is true not just of *large* scientific databases (more than a Terabyte in size) but even smaller databases that make extensive use of the full complement of database management system (DBMS) features for performance and user convenience.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems – *Distributed Databases*.

General Terms

Your general terms must be any of the following 16 designated terms: Management, Performance.

Keywords

Databases, Cloud, Scientific Databases, Data in the Cloud. Cloud Computing

1. INTRODUCTION

The hosting of large digital archives of scientific data for indefinite online access by a large and worldwide user community is a daunting undertaking for most academic institutions and scientific laboratories, especially because it is inevitably under-budgeted in the project planning. Any dent that cloud computing services [2] can make in this regard would be most welcome.

As a case in point, the main site of the Sloan Digital Sky Survey's Catalog Archive Server (CAS) [3] at FermiLab is hosted on a cluster of 25 commodity class servers connected to over 100 TB of storage, with 2.5-3 FTE operational resources committed to supporting the archive and maintaining high availability. This includes support for multiple copies and versions of what is essentially a 5 TB database.

The CAS is essentially a Microsoft SQL Server DBMS containing the SDSS Science Archive data. The Science Archive contains essentially all the reduced data and science parameters extracted from the raw (binary) data obtained at the telescope. These data and parameters are then loaded into the SQL Server database using a semi-automated loading pipeline called sqlLoader [4]. There are two views of the Science Archive data. In addition to the CAS, there is also a Data Archive Server (DAS) [5] analogous to the CAS that provides users access to the raw (file) data in a binary format popular among astronomers. Users can download tarballs of the file data from the DAS using wget and rsync.

The enormous size of the SDSS Science Archive (information content larger than the US Library of Congress and a thousand-fold increase in data over all previous archives combined) made it completely unusable only as a file-based archive. The ability to search the data quickly and extract only the desired parameters was absolutely essential in order to deliver the true potential of a dataset unprecedented in size and richness. The SDSS collaboration decided at the outset to extract the science data into a DBMS and make it available through the CAS. In addition to the data tables, the CAS contains extensive usability and performance enhancements that make its schema quite complex and difficult to port to other DBMS platforms. This complexity is also a big obstacle for migrating the database to current cloud platforms.

Although the production configuration of the CAS at FermiLab deploys multiple copies of a given SDSS database (e.g. DR6) on different CAS servers for load balancing, for the purposes of the tests described here, we are comparing a single DR6 database on a

single dedicated (and isolated) server to compare with the cloud instance of the same data. At least to start with, we wanted to see how one of our high end servers would stack up against a cloud implementation.

To date, we have attempted to migrate the DR6 data to two commercial cloud computing services that provide SQL Server database hosting within the cloud – Amazon Elastic Cloud Computing (EC2) and Microsoft SQL Azure. This paper describes our experiences with each of these cloud services.

2. SDSS DATA ON AMAZON EC2

The primary motivation for deploying SDSS data in the cloud was to compare cost-effectiveness and performance of hosting and accessing the data in the cloud. Although databases have been deployed in the EC2 cloud before, ours was the first attempt to put a reasonably large SQL Server database in the cloud. In fact, this attempt got off the ground when Amazon approached us and said they were interested in hosting SDSS as one of the public datasets on the Amazon Web Services (AWS) site.

Amazon EC2 (http://aws.amazon.com/ec2/) is a Web service that provides resizable compute capacity in the cloud. EC2 is billed as a true virtual environment that provides a Web services interface to:

- launch instances of a variety of operating systems,
- load them with custom application environments,
- manage your network's access permissions, and
- run your image (see AMI below) with as many systems as you like.

Amazon Elastic Block Store (EBS) provides block level storage volumes for use with Amazon EC2 instances. The storage persists independently of the life of the instance. EC2 instances and EBS volumes are created and administered from the AWS Management Console (http://aws.amazon.com/console/), using your Amazon account (if you have an account on amazon.com for shopping, you can use that account and have service charges billed to your Amazon credit card).

The AWS model is that the database is stored as a "snapshot" (i.e. a copy taken at a point in time) available on the AWS site, and if it is a public (free) dataset like SDSS, it is advertised on the AWS blog (http://aws.typepad.com/). Although snapshots are supposedly differential backups, they can also be used to instantiate new EBS volumes. Anyone can then pull the snapshot into their AWS account to create a running instance (at this point they start incurring AWS charges). Multiple instances have to be deployed manually. Since deploying one instance entails a number of steps (Figure 1), this can become time-consuming and cumbersome.

In order to create a running instance of a SQL Server database on EC2, you first have to create the storage you need for the database by creating an EBS volume. This is done by instantiating your snapshot as an EBS volume of the required size (we selected 200 GB as the volume size, which is a "big" volume). Then you select a SQL Server 2005 (now 2008 must be available too) Amazon Machine Image (AMI) for the dataset snapshot available on AWS, and create an instance of this AMI. Next you attach this instance to the EBS volume, which creates a running instance. Finally, you create an elastic IP for this instance so the outside world can connect to it. It is not possible to set up a SQL Server cluster within the cloud (i.e. interconnect multiple instances), as far as we know (this may be possible with Amazon Virtual Private Cloud).

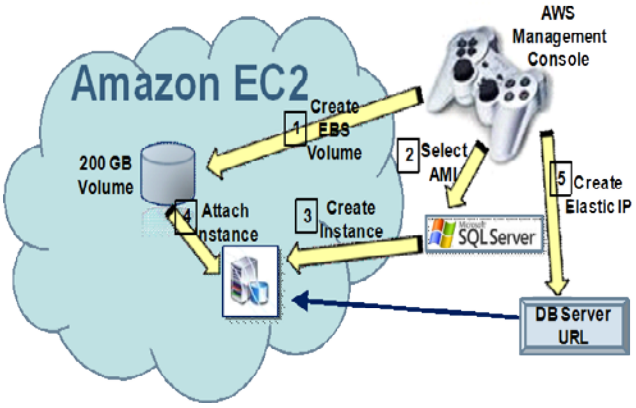

Figure 1. Steps needed to create an Amazon EC2 instance of the 100 GB SDSS subset database (numbered from 1 to 5). These steps must be repeated for each instance in the cloud.

As mentioned above, the full SDSS (Data Release 7) dataset is 5 TB in size. Amazon EC2 is limited to a maximum of 1 TB per instance for the size of database they can host. So right off the bat, it was clear that they would not be able to host the full SDSS database, since we did not have an easy way of splitting up the dataset into 1 TB slices as yet. Although we will have the ability to partition the dataset in the future, presumably the layer to reduce these to one logical dataset would have to be outside the cloud. Regardless, we are still interested in investigating the current cloud environment to see how easy it is to deploy a database to it and how it performs, how usable it is, etc. Indeed, we anticipate that it should be possible in the near future to deploy the whole SDSS database to AWS and other cloud environments.

In order to have a dataset that was large enough to provide a realistic test of performance and scalability, but also not be too large so that it would be expensive and time consuming to run our tests, we chose a 100 GB subset of the SDSS DR6 database (the full DR6 database is about 3.5 TB in size). This ~ $1/35^{th}$ size subset is generated by restricting the sky area covered by the data to a small part of the total sky coverage for SDSS, i.e. a few 100 sq degrees rather than thousands of square degrees.

2.1 Migrating the Data

With most of the other databases on AWS, the assumption is that users will set up their own database first and *then* import the data into it. In our case, since we had a pre-existing (and large) database with a complex schema, it made much more sense for us to migrate the database in one piece to the EC2 virtual server. There are two ways to do this – either with a SQL Server backup of the database at the source and a corresponding restore in the cloud, or by detaching the database and copying the data file(s) to the cloud volume. For the AWS snapshot model, the latter was the more suitable option, so we chose that.

2.2 Performance Testing

We have a 35 query test suite that we routinely use to test and benchmark SDSS servers [6]. The queries are all encoded in a single SQL Server stored procedure – spTestQueries – that can be set up to run the whole suite as many times as desired. The

queries range from spatial (radial "cone" search) queries to complex joins between multiple tables. For each query executed, three performance counters are measured – the elapsed time, the CPU seconds and the physical IO.

Although the production CAS site at FermiLab contains 25 database servers, each server has one copy of a given SDSS database, and load-balancing and performance is achieved by segregating different workloads among different servers. As such, we only need to compare the performance of a single instance/server of the database inside and outside the cloud, at least to a first approximation.

The test query suite generally assumes that the server it is running on is isolated and offline, and also that certain performance enhancements are in installed, foremost among them being the Hierarchical Triangular Mesh (HTM) spatial indexing library [7] that provides fast (O(log N)) spatial searching. The library is implemented in C# and used the SQL-CLR (Common Language Runtime) binding along with some SQL glue functions.

Figure 2 shows the comparison between running this test suite on our GrayWulf [8][9] server and on the instance of the database on EC2. Only the query elapsed time (in seconds) is shown in the plot, and the differences are large enough that a logarithmic scale was required to plot the times. The EC2 elapsed times are on average an order of magnitude larger than the ones we obtained on the GrayWulf (single) server instance. The difference could be due a number of factors, such as the database settings on the EC2 server (memory, recovery model, tempdb size etc.) as well as the disk speeds.

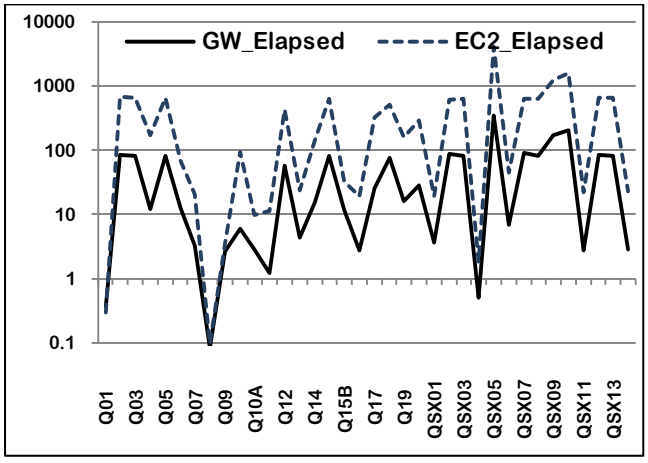

Figure 2. Comparison of query elapsed time for the 100-GB SDSS subset on our GrayWulf server (GW) and EC2. We ran 35 test queries (alternate query numbers are shown) from our test query suite on each database. The elapsed times are in seconds.

The purpose of this comparison is not to draw a definitive conclusion about the relative performance of the two types of instances, but rather to indicate that the performance in the cloud can be disappointing unless it can be tuned properly. We only used the default settings for the most part on EC2, and it might have been possible (assuming the privileges were available) to tweak the performance settings to our benefit.

2.3 Data Access

The public SDSS dataset on AWS was meant to serve two kinds of users on:

a) People who currently use the SDSS CAS

b) General AWS users who were interested in the data

For a), we needed to be able to replicate the same services that SDSS currently has, but using a SQL Server instance on EC2 and connecting to it with the elastic IP resulting from the process described in Figure 1. This should in theory work fine, although we were unable to make the connection work during our tests. Although we were able to log in to the EC2 server fine using a Windows remote desktop client, we could not connect to the elastic IP using a SQL OLEDB (a Microsoft protocol that allows applications to connect to SQL Server using a connection string) connection, and hence could not connect a SkyServer Web interface [6] to it.

For b), users should have everything they need on the AWS public dataset page for SDSS, but here was the rub: it was quite a complex set of maneuvers that a potential user (by user here we mean someone who wants to provide access to the SDSS database, not an end-user; so for example JHU would be a user) would have to execute is a quite daunting (see Figure 3). The most difficult part by far is "porting the SDSS data to another database (platform)". SQL Server EC2 instances should not be difficult to create and provide access to. (As an interesting aside, AWS also made the SDSS public dataset available as a LINUX snapshot, which did not make sense to us since SQL Server cannot run on LINUX).

> The data set takes the form of a Microsoft SQL Server MDF file. Once you have created your EBS volume and attached it to your Windows EC2 instance, you can access the data using SQL Server Enterprise Manager or SQL Server Management Studio. The SDSS makes use of stored procedures, user defined functions, and a spatial indexing library, so porting it to another database would be a fairly complex undertaking.

Figure 3. A description of the procedure required to use the SDSS public dataset as provided on the AWS blog.

2.3.1 Cost of Data Access

Another important aspect of using EC2 (or any other cloud) instances of datasets like SDSS is the cost-effectiveness of the data access. We do not have any useful information to contribute on this as yet, partly because we were unable to get the Web interface connection to work. We incurred charges of nearly $500 for our experiments, without actually providing any remote data access. Basically, all we did was create the instance and run our performance test queries on it. The duration of the tests was a few weeks all told, and the database was idle for part of the time while we were busy with other commitments. Most of the charges were for "EC2 running large Windows instance with Authentication Services". The EBS storage and other miscellaneous charges were negligible by comparison, even though we had a 200 GB EBS volume in use for all that time. This would indicate that licensing costs for third party software (in this case Windows Server and SQL Server) is the dominant factor. If that is indeed the case, this could potentially make it infeasible to put science datasets like SDSS in the cloud.

3. SDSS DATA ON MICROSOFT SQL AZURE

This is really a work in progress. At this time we are not able to report on performance specifics, but hopefully soon. In the meantime, it is instructive to look at some of the difficulties we have experienced in migrating the data to the Azure Cloud.

3.1 Migrating the Data

Although the creation of a SQL Azure project yields a SQL Server instance and an IP address for it, there currently appears no way to directly move a database "as is" or *en* masse into the cloud, even if the database is first upgraded to the proper SQL Server version (in this case SQL Server 2008). The SQL Azure instructions and labs offer two options for moving data into the cloud: using the built-in database scripting facility in SQL Server 2008 to script the schema and data export, and using the bulk copy utility (BCP) to copy data from the on-premise database to Azure. Given the nature of the SDSS schema and data, for example the heavy reliance on indexes, stored procedures and user-defined functions as well as the CLR (common language runtime) assemblies used for the spatial indexing library, both of these options are fraught with problems. Using the scripting option, we ran out of memory while generating the script! In fact, even if the script had been generated successfully, according to the instructions it needs to be then edited manually to remove features not supported by SQL Azure.

3.1.1 SQL Azure Migration Wizard

There is, in fact, a third option that the instructions do not explicitly mention (or if they did, we missed it!). At the moment, we are using the SQL Azure Migration Wizard (SAMW - http://sqlazuremw.codeplex) to move the data into Azure. SAMW actually does an admirable job of automating the migration task. However, as we shall see, this does not eliminate our problems.

As in the case of the first migration option (using SQL Server built-in scripts), SAMW removes any schema elements that are not supported by Azure. Many functions and stored procedures in the SDSS schema do not get ported to the Azure copy of the database. This makes it very difficult to compare the deployment to say the AWS version. One has to go through the voluminous SAMW trace (Figure 4) to find all the errors it encountered. Some of the unsupported features that prevent these functions and stored procedures from being migrated are:

- References to other databases – this is a minor inconvenience which we can work around by simply deleting references to other databases in most cases. In some cases, it is not so easy to work around it, for example where it prevents the use of the command shell from within SQL Server (the shell must be invoked via the *master* database). This means that we cannot run command (including SQL) script files from SQL Server. However, for now we are ignoring these issues and soldiering ahead.
- Global temp objects – this prevents the test query stored procedure (spTestQueries) from being migrated in its original form. The procedure uses global temp variables to record the performance metrics. A workaround for this is not trivial because this is one of the main functions of the test script.
- T-SQL directives – these are special directives in the SQL Server SQL dialect (Transact-SQL or T-SQL for short) to control how certain commands or procedures are executed, e.g., to set the level of parallelism. These are mostly performance related, but for admin tasks rather than user queries, so they can be ignored if necessary.
- Built-in T-SQL functions – these are also mostly in admin functions and procedures, so not a big concern for now.
- SQL-CLR function bindings – this is a big one, because this means we cannot use our HTM library to speed up the spatial searches.
- Deprecated features – since our SQL code was mostly developed on an earlier version (SQL Server 2000), it contains some features deprecated in SQL Server 2008. These are not supported in Azure. We will have to remove them, which is not a major problem since there are very few such cases, and it is good to remove them anyway in the long run.

The bottom line is that migrating the data to the SQL Azure cloud currently involves stripping out several features that will at the very least impact performance of our database, and could potentially make some aspects of it unusable.

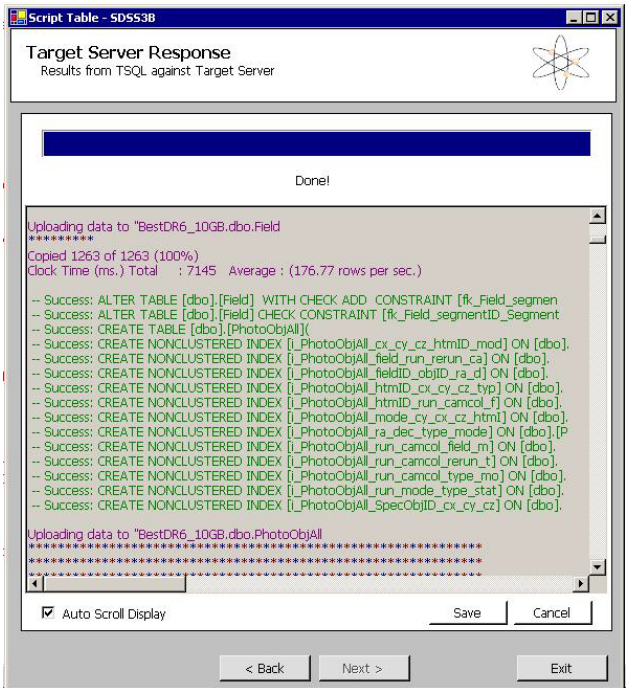

Figure 4. Screen shot of SQL Azure Migration Wizard session showing migration trace for SDSS DR6 10 GB subset. Here "Target Server" is the Azure database server. The wizard is running on our local SDSS server at JHU.

3.2 Performance Testing

Since this is a 10 GB subset (the actual size is actually closer to 6 GB), the performance test results will be much more difficult to compare with the 100 GB and full size databases. However, we aim to run the test query suite on the same database in and out of the cloud. The major problem here though is the anticipated modifications that will be needed during the migration process due to the features not currently supported by SQL Azure (see above). If changes are made to settings which affect the

performance in a significant way, then it will not be possible to obtain a meaningful performance benchmark. This is indeed emblematic of the difficulties in deploying and benchmarking large databases in the cloud at the moment.

3.3 Data Access

The IP-address that we created for the SQL Azure server allows us to connect to the server, and we have been able to connect to the Azure instance of the DR6 subset in two ways:

1. We can hook up a SkyServer [6] client (Web interface) from outside the cloud using a SQL OLEDB connection. Although the full functionality of SkyServer queries is not available in the cloud version (due to the subset of supported features as mentioned above), we have been able to run a good fraction of the SDSS sample queries.

2. We can also connect to the Azure instance as a database engine from a remote SQL Server Management Studio client using the admin user login that Azure provides. This allows us to configure the database just as we would a local database. We are using this mode of access to work around the Azure limitations, by creating kosher versions of functions and stored procedures as necessary.

We will not have the data access costs until we address all the migration issues listed in §3.1. For these tests we purchased a 10 GB developers' package which costs ~ $100/month and includes a certain amount of free data transfers in addition to the 10 GB storage and licensing costs.

4. CONCLUSIONS

We have so far migrated a 100 GB subset of a large astronomical database – the Sloan Digital Sky Survey science archive – to the Amazon EC2 cloud. EC2 has a size limit of 1 TB per instance, so it was not possible for us to migrate the whole SDSS database (several TB) to it and perform a full test. After much help from the Amazon experts, we were able to install an instance of the SDSS data in EC2 and run our test suite of 35 test queries on it. With only the default settings and a single EC2 instance, we found the query performance to be an order of magnitude slower than our on-premise GrayWulf server. This was indicative of the need to either tine the EC2 performance settings or create more than one instance to get better performance. Creating an EC2 instance was a multi-step process that needed to be followed for each instance. After successfully creating an instance and testing its performance, we were unable to access the instance with the public IP-address (elastic IP) generated using the AWS instructions. As such, the instance was not accessible from the outside world.

We are in the process of migrating a much smaller (10 GB) subset of the same dataset to the Microsoft SQL Azure cloud (10 GB is the current size limit for Windows/SQL Azure). The challenge with SQL Azure – other than the 10 GB size limit which is really too small to do any realistic tests – is that direct migration of the data is not possible at the moment, since SQL Azure supports a subset of database features and hence database migration must be scripted or done using special purpose utilities. Even with these tools, the version of the database in the cloud is significantly altered and cannot support the full functionality of the original database. It certainly cannot match the performance of the original version. In fact it is not even possible to measure the performance of the migrated database in the same way as the original so as to make a meaningful comparison.

At this time, it is not possible to migrate and access a scientific SQL Server database in the Amazon and SQL Azure clouds, at least based on our admittedly incomplete experiments. Even as the limits on the database size expand in the near future, there are problems with migrating the data itself, and then providing the type of performance and access possible desired. Beyond that, the licensing costs for software used in the cloud could become a significant issue. We hope to have a more positive report soon as we continue to explore migrating science data to the cloud.

5. ACKNOWLEDGMENTS

Our thanks to the Amazon Web Services folks, in particular Santiago Alonso Lord, Deepak Singh and Jeffrey Barr (who maintains the AWS blog), for hosting a public dataset and for all their patient help in setting up and transferring SDSS data to EC2. Also thanks to Roger Barga (Microsoft Research) for his help with migrating data to SQL Azure, and pointing us to the SQL Azure Migration Wizard.

6. REFERENCES

[1] Thakar, A.R. 2008: "The Sloan Digital Sky Survey: Drinking from the Fire Hose", Computing in Science and Engineering, 10, 1 (Jan/Feb 2008), 9.

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. Above the Clouds: A Berkeley View of Cloud computing. Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, Feb. 10, 2009.

[3] Thakar, A.R., Szalay, A.S., Fekete, G., and Gray, J. 2008: "The Catalog Archive Server Database Management System", Computing in Science and Engineering, 10, 1 (Jan/Feb 2008), 30.

[4] Szalay, A.S., Thakar, A.R.,, and Gray, J. 2008: "The sqlLoader Data Loading Pipeline", Computing in Science and Engineering, 10, 1 (Jan/Feb 2008), 38.

[5] Neilsen, Jr., E. H.,. 2008: "The Sloan Digital Sky Survey Data Archive Server", Computing in Science and Engineering, 10, 1 (Jan/Feb 2008), 13.

[6] J. Gray, A.S. Szalay, A. Thakar, P. Kunszt, C. Stoughton, D. Slutz, J. vandenBerg. "Data Mining the SDSS SkyServer Database," *Distributed Data & Structures 4: Records of the 4th International Meeting*, pp 189-210 W. Litwin, G. Levy (eds), Paris France March 2002, Carleton Scientific 2003, ISBN 1-894145-13-5, also MSR-TR-2002-01, Jan. 2002: http://research.microsoft.com/pubs/64558/tr-2002-01.pdf.

[7] Szalay,A., Gray, J., Fekete, G., Kunszt, P., Kukol, P., and Thakar, A., "Indexing the Sphere with the Hierarchical Triangular Mesh", Microsoft Technical Report 2005, http://research.microsoft.com/apps/pubs/default.aspx?id=64531

[8] Szalay, A.S. et al. 2008, "GrayWulf: Scalable Clustered Architecture for Data Intensive Computing" Microsoft Technical Report MSR-TR-2008-187.

[9] Simmhan, Y. et al. 2008, "GrayWulf: Scalable Software Architecture for Data Intensive Computing" Microsoft Technical Report MSR-TR-2008-186.