Scientific Programming 19 (2011) 147–159
DOI 10.3233/SPR-2011-0325
IOS Press

Large science databases – are cloud services ready for them?

Ani Thakar [a,*], Alex Szalay [a], Ken Church [b] and Andreas Terzis [c]

[a] *Department of Physics and Astronomy and the Institute for Data Intensive Engineering and Science, The Johns Hopkins University, Baltimore, MD, USA*

[b] *Human Language Technology Center of Excellence and IDIES, The Johns Hopkins University, Baltimore, MD, USA*

[c] *Department of Computer Science and IDIES, The Johns Hopkins University, Baltimore, MD, USA*

Abstract. We report on attempts to put an astronomical database – the Sloan Digital Sky Survey science archive – in the cloud. We find that it is very frustrating to impossible at this time to migrate a 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 that would degrade performance and/or make the data unusable. Preliminary performance comparisons show a large performance discrepancy with the Amazon cloud version of the SDSS database. These difficulties suggest that much work and coordination needs to occur between cloud service providers and their potential clients before science databases – not just large ones but even smaller databases that make extensive use of advanced database features for performance and usability – can successfully and effectively be deployed in the cloud. We describe a powerful new computational instrument that we are developing in the interim – the Data-Scope – that will enable fast and efficient analysis of the largest (petabyte scale) scientific datasets.

Keywords: Databases, cloud computing

1. Introduction

The hosting of large digital archives of scientific data, like the Sloan Digital Sky Survey (SDSS) science archive [17], 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. The economies of scale and on-demand provisioning enabled by cloud computing services [1] can potentially make a significant dent in the operational budgets of large scientific archive projects that would be most welcome. This will become increasingly important as we go to ever larger data sizes – from terabytes to petabytes – over the next few years.

[*]Corresponding author: Ani Thakar, Department of Physics and Astronomy and the Institute for Data Intensive Engineering and Science (IDIES), The Johns Hopkins University, 3701 San Martin Drive, Baltimore, MD 21218-2695, USA. Tel.: +1 410 516 4850; Fax: +1 410 516 4477; E-mail: thakar@jhu.edu.

This is the primary motivation for the cloud experiments described below. As a case in point, the main site of the SDSS Catalog Archive Server (CAS) [18] at FermiLab is hosted on a cluster of 25 commodity class servers connected to over 100 TB of storage, with 2.5–3 FTE (full-time equivalent) 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 SQL Loader [15]. There are two views of the Science Archive data. In addition to the CAS, there is also a Data Archive Server (DAS) [6] 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., Data Release 6 (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. We describe below our experiences with each of these cloud services.

Due to the difficulties we encountered with moving small to modest sized scientific datasets into these cloud services, we have concluded that moving the largest current and upcoming datasets to commercial cloud services for efficient data analysis is not a viable option at the present time. Accordingly, we describe the Data-Scope, our attempt to build an instrument designed specifically for scientific data analysis on the largest scales. The Data-Scope will meet the specific needs of analyzing terabytes of scientific data at a time with commodity components to reduce cost. It will seek to maximize sequential disk throughput while minimizing power consumption so as to avoid the "power wall" that large-scale computing is about to hit. The two-tier architecture of the Data-Scope combines the best state-of-the-art technologies to achieve maximum data throughput at minimal cost and power usage.

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 (Fig. 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

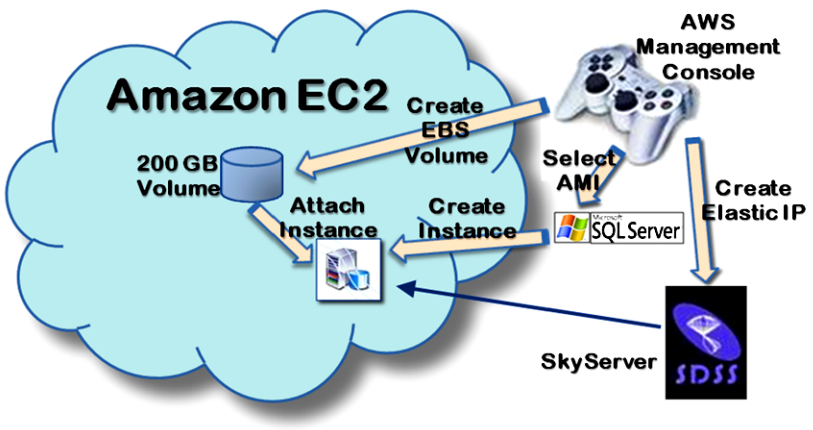


Fig. 1. Steps needed to create an Amazon EC2 instance of the 100 GB SDSS subset database (numbered from 1 to 5). These steps must be repeated for each instance in the cloud.

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).

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 $\sim 1/35$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 [4]. 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 work-

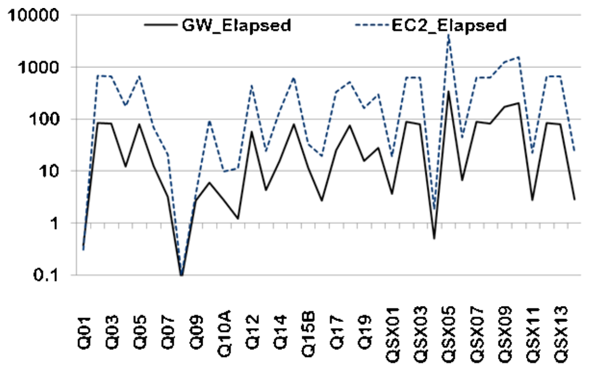


Fig. 2. Comparison of query elapsed time for the 100-GB SDSS subset on our GrayWulf server (GW) and EC2. We ran 35 test queries (alternate query numbers are shown) from our test query suite on each database. The elapsed times are in seconds.

loads 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 [14] 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 [10] 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.

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 Fig. 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 [4] 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 Fig. 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).

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

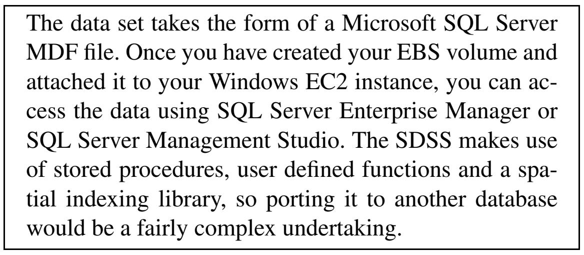

The data set takes the form of a Microsoft SQL Server MDF file. Once you have created your EBS volume and attached it to your Windows EC2 instance, you can access the data using SQL Server Enterprise Manager or SQL Server Management Studio. The SDSS makes use of stored procedures, user defined functions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

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

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

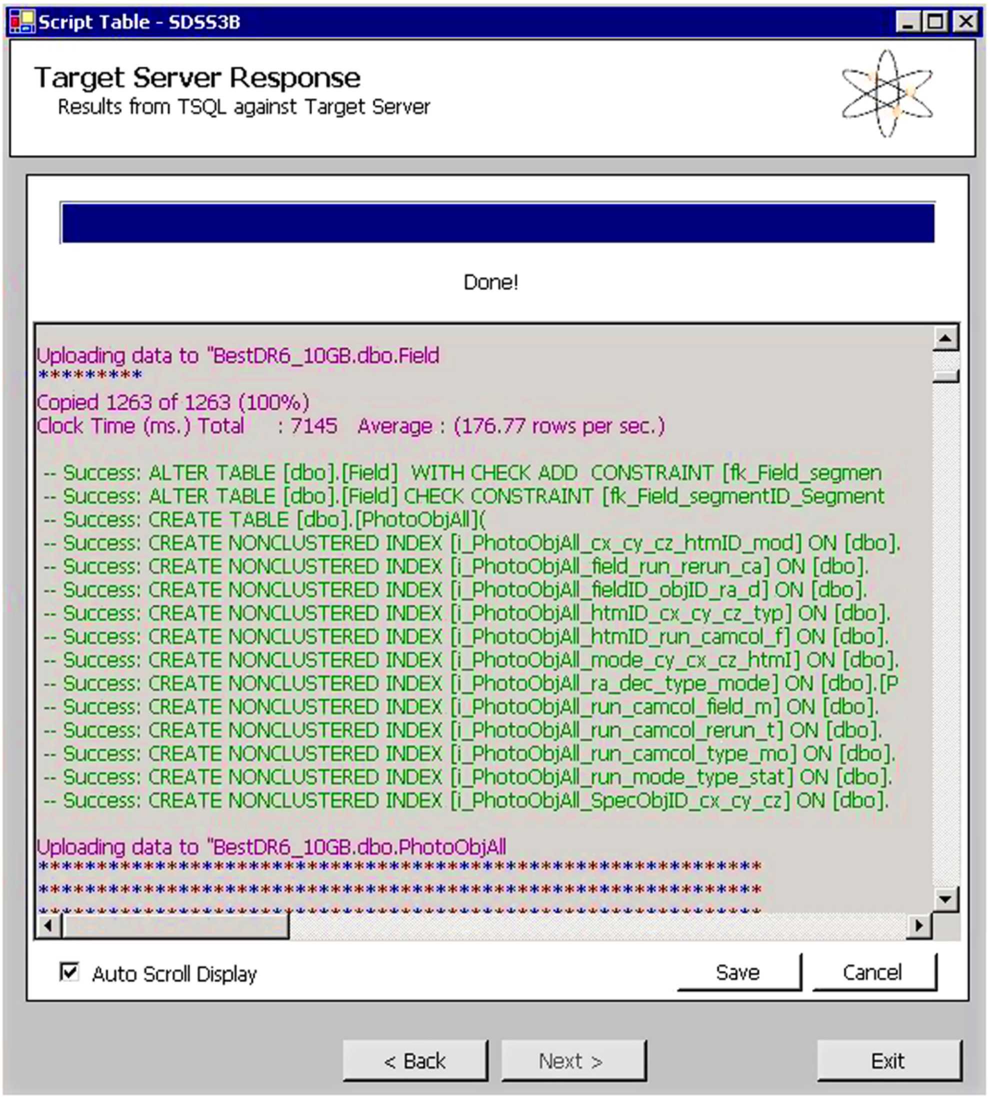
Although the creation of a SQL Azure project yields a SQL Server instance and an IP address for it, there currently appears no way to directly move a database "as is" or *en* masse into the cloud, even if the database is first upgraded to the proper SQL Server version (in this case SQL Server 2008). 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 (Fig. 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.



```
Uploading data to "BestDR6_10GB.dbo.Field
*********
Copied 1263 of 1263 (100%)
Clock Time (ms.) Total    : 7145   Average : (176.77 rows per sec.)

 -- Success: ALTER TABLE [dbo].[Field]  WITH CHECK ADD  CONSTRAINT [fk_Field_segmen
 -- Success: ALTER TABLE [dbo].[Field] CHECK CONSTRAINT [fk_Field_segmentID_Segment
 -- Success: CREATE TABLE [dbo].[PhotoObjAll](
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_cx_cy_cz_htmID_mod] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_field_run_rerun_ca] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_fieldID_objID_ra_d] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_htmID_cx_cy_cz_typ] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_htmID_run_camcol_f] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_mode_cy_cx_cz_htmI] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_ra_dec_type_mode] ON [dbo].[P
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_run_camcol_field_m] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_run_camcol_rerun_t] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_run_camcol_type_mo] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_run_mode_type_stat] ON [dbo].
 -- Success: CREATE NONCLUSTERED INDEX [i_PhotoObjAll_SpecObjID_cx_cy_cz] ON [dbo].

Uploading data to "BestDR6_10GB.dbo.PhotoObjAll
****************************************************************
****************************************************************
```

Fig. 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.

- 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.

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 [4] 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 cannot begin to assess the data access costs until we address all the migration and functionality issues listed above. 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.

The data movement issues for both Amazon and Microsoft clouds are largely an artifact of migrating a (large) dataset *that already exists outside the cloud* into the cloud. When cloud services are fully integrated, we expect that ingesting the data into the database(s) will also be done inside the cloud, so this will not be a major issue. Essentially, the entire catalog archive server will have to be implemented in the cloud from the outset to make the most effective use of cloud services.

4. Data-Scope

From the experiences described above with moving data in and out of the Amazon EC2 and SQL Azure cloud services, it is clear that commercial cloud computing platforms are not yet the solution to the problems posed by the largest scientific datasets. Apart from the logistical problems with moving and accessing the data, the economics are also not favorable as yet. The data storage, and especially the data movement and access fees are excessive compared to purchasing physical disks, the IO performance they offer is substantially lower (∼20 MBps), and the amount of provided disk space is often woefully inadequate (e.g., ∼50 GB per Azure instance). The cloud may be an attractive alternative for modest sized datasets that can either be generated within the cloud or that can be migrated into the cloud without too many technical hurdles. The data size, schema complexity, advanced feature requirements, and performance demands associated with the largest scientific datasets make it infeasible to port them to commercial cloud computing platforms at the present time.

We believe that there is a vacuum today in data-intensive scientific computation regime, similar to the one that led to the development of the BeoWulf cluster: an inexpensive yet efficient template for data intensive computing in academic environments based on commodity components. The Data-Scope instrument that we are developing aims to fill this gap. This instrument is intended to provide an immediate solution to the challenges posed by the largest scientific datasets, but we also expect that its development, use and testing (e.g., benchmarking) will inform future adaptations of commercial platforms for data intensive science at the highest scales. Data-Scope resources will be free to use for scientific data experiments subject to availability.

4.1. *The design concept*

The Data-Scope we are building at the Johns Hopkins University Institute for Data Intensive Engineering and Science (IDIES) is an instrument optimized for analyzing petabytes of data in an academic setting where cost and performance considerations dominate ease of management and security. The Data-Scope will form a template for other institutions facing similar challenges, at least until cloud services catch up to the needs of the largest datasets in science.

The following requirements guide the Data-Scope's design:

(a) *Provide at least* 5 *petabytes of storage, with a safe redundancy built in.*
(b) *Keep the ratio of total system to raw disk costs as low as possible.*
(c) *Provide maximal sequential throughput, approaching the aggregate disk speed.*

(d) *Allow streaming data analyses on par with data throughput (i.e.,* 100 *s of TFlops).*

(e) *Maintain total power requirements as low as possible.*

This ordered list maps well onto the wish list of most academic institutions. The tradeoff is in some aspects of fault tolerance, the level *of automation in data movement and recovery and a certain complexity in programming convenience* since the high stream-processing throughput at a low power is achieved by using Graphics Processing Units (GPUs). These tradeoffs, combined with maximal use of state of the art commodity components, will allow us to build a unique system, which can perform large data analysis tasks simply not otherwise possible. The Data-Scope will enable JHU scientists and their collaborators to:

- Bring their 100 TB+ data sets to the instrument, analyze them for several months at phenomenal data rates and take their results "home".
- Create several long-term, robust and high performance services around data sets in the 10–200 TB range, and turn them into major public resources.
- Explore new kinds of collaborative research in which even the shared, temporary resources can be in the hundreds of terabytes and kept alive for several months.
- Explore new data-intensive computational and data analysis paradigms enabled by the intersection of several technologies (HPC, Hadoop, GPU) and toolkits like CUDA[1]-SQL and MPI-DB.

In the paragraphs that follow we describe the Data-Scope's hardware and software designs. The hardware layout is shown in Fig. 5.

4.2. The hardware design

The driving goal behind the Data-Scope design is to maximize stream processing throughput over TB size datasets while using commodity components to keep acquisition and maintenance costs low. Performing the first pass over the data directly on the servers' PCIe backplane is significantly faster than serving the data from a shared network file server to multiple compute servers. This first pass commonly reduces the data significantly, allowing one to share the results over the network without losing performance. Furthermore, providing substantial GPU capabilities on the same server enables us to avoid moving too much data across the network as it would be done if the GPUs were in a separate cluster. Since the Data-Scope's aim is providing *large* amounts of *cheap* and *fast* storage, its design must begin with the choice of hard disks. There are no disks that satisfy all three criteria. In order to balance these three requirements we decided to divide the instrument into two layers: *performance* and *storage*. Each layer satisfies two of the criteria, while compromising on the third. Performance Servers will have high speed and inexpensive SATA drives, but compromise on capacity: Samsung Spinpoint HD103SJ 1 TB, 150 MB/s (see [12], verified by our own measurements). The Storage Servers will have larger yet cheaper SATA disks but with lower throughput: Samsung Spinpoint HD203WI 2 TB, 110 MB/s. The storage layer has 1.5× more disk space to allow for data staging and replication to and from the performance layer.

The rest of the design focuses on maintaining the advantages from these two choices. In the *performance layer* we will ensure that the achievable aggregate data throughput remains close to the theoretical maximum, which is equal to the aggregate sequential IO speed of all the disks. As said before, we achieve this level of performance by transferring data from the disks over the servers' local PCIe interconnects rather than slower network connections. Furthermore, each disk is connected to a separate controller port and we use only 8-port controllers to avoid saturating the controller. We will use the new LSI 9200-series disk controllers, which provide 6 Gbps SATA ports and a very high throughput (we have measured the saturation throughput of the LS92111-8i to be 1346 MB/s). Each performance server will also have four high-speed solid-state disks (OCZ-Vertex2 120 GB, 250 MB/s read, 190 MB/s write) to be used as an intermediate storage tier for temporary storage and caching for random access patterns [9].

The performance server will use a SuperMicro SC846A chassis, with 24 hot-swap disk bays, four internal SSDs, and two GTX480 Fermi-based NVIDIA graphics cards, with 500 GPU cores each, offering an excellent price-performance for floating point operations at an estimated 3 teraflops per card. The Fermi-based TESLA 2050 has not been announced yet, we will reconsider if it provides a better price performance as the project begins. We have built a prototype system according to these specs and it performs as expected.

[1]CUDA stands for Compute Unified Device Architecture and is a parallel computing architecture developed by NVIDIA that is optimized for GPUs (http://www.nvidia.com/object/what_is_cuda_new.html).

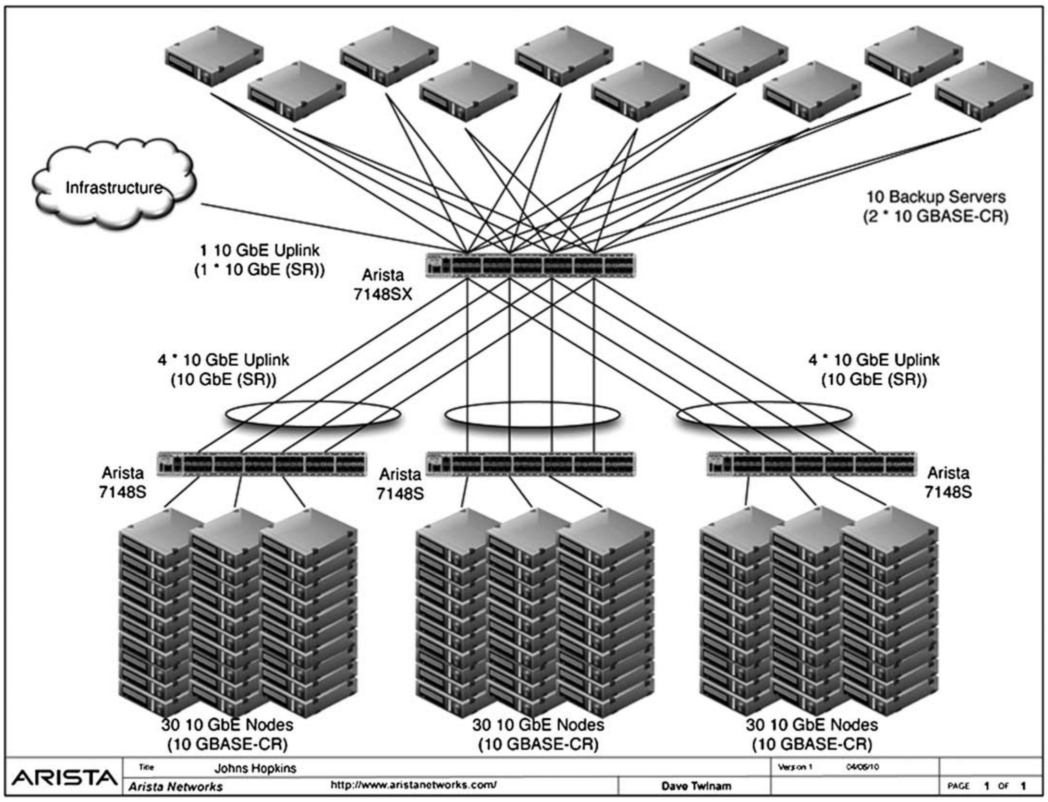


Fig. 5. The network diagram for the Data-Scope.

In the *storage layer* we maximize capacity while keeping acquisition costs low. To do so we amortize the motherboard and disk controllers among as many disks as possible, using backplanes with SATA expanders while still retaining enough disk bandwidth per server for efficient data replication and recovery tasks. We will use locally attached disks, thus keeping both performance and costs reasonable. All disks are hot-swappable, making replacements simple. A storage node will consist of 3 SuperMicro SC847 chassis, one holding the motherboard and 36 disks, with the other two holding 45 disks each, for a total of 126 drives with a total storage capacity of 252 TB. On the storage servers we will use one LSI-9211-8i controller to drive the backplane of the 36 disks, connecting 2×4 SATA ports to the 36 drives, through the backplane's port multiplier. The two external disk boxes are connected to a pair of LSI 9200-8e controllers, with 2×4 ports each, but the cards and boxes are cross-wired (one 4-port cable from each card to each box), for redundancy in case of a controller failure, as the split backplanes automatically revert to the good controller. The IO will be limited by the saturation point of the controllers and the backplanes, estimated to be approximately 3.6 GB/s.

Both servers use the same dual socket SuperMicro IPMI motherboard (X8DAH+FO) with 7 PCIeGen2 slots. The CPU is the cheapest 4-core Westmere, but we will be able to upgrade to faster dual 6-cores in the future, as prices drop. In our prototype we tried to saturate this motherboard: we exceeded a sequential throughput of 5 GB/s with no saturation seen.

The network interconnect is 10 GbE. Three 7148S switches from Arista Networks' are used at the "Top of the Rack" (TOR), and a high performance 7148SX switch is used for the "core" and the storage servers. The TOR switches each have four links aggregated to the core for a 40 Gbps throughput. We deploy Chelsio NICs, single port on the performance servers and dual port on the storage side.

4.2.1. Hardware capabilities

The Data-Scope will consist of 90 performance and 12 storage servers. Table 1 shows the aggregate properties of the full instrument. *The total disk capacity will exceed* 5 *PB, with* 3 *PB in the storage and* 2.2 *PB in the performance layer. The peak aggregate sequential*

Table 1

Summary of the Data-Scope properties for single servers and for the whole system consisting of performance (P) and storage (S) servers

	1P	1S	90P	12S	Full	
Servers	1.0	1.0	90	12	102	
Rack units	4.0	12.0	360	144	504	
Capacity	24.0	252.0	2160	3024	5184	TB
Price	8.5	22.8	766	274	1040	$K
Power	1.0	1.9	94	23	116	kW
GPU	6.0	0.0	540	0	540	TF
Sequential IO	4.6	3.8	414	45	459	GBps
Network bandwidth	10.0	20.0	900	240	1140	Gbps

IO performance is projected to be 459 *GB/s, and the peak GPU floating point performance will be* 540 *TF.* This compares rather favorably with other HPC systems. For example, the Oak Ridge Jaguar system, the world's fastest scientific computer, has 240 GB/s peak IO on its 5 PB Spider file system [11]. The total power consumption is only 116 kW, a fraction of typical HPC systems, and a factor of 3 better per PB of storage capacity than its predecessor, the GrayWulf. The total cost of the parts is $1.04M. The projected cost of the assembly + racks is $46K, and the whole networking setup is $114K, for a total projected hardware cost of about $1.2M. In the budget we reserved $200K for contingency and spare parts. The system is expected to fit into 12 racks.

4.3. Data ingestion and recovery strategies

The storage servers are designed for two purposes, *Data Replication and Recovery,* incremental and full dataset copies and restores (large and small), and *Import/Export of Large Datasets*, where users show up with a couple of boxes of disks and should be able to start experiments within hours, and keep their data online over the lifetime of the experiment (e.g., months).

Individual disk failures at the expected standard rate of about 3%/year are not expected to cause much of a problem, for the performance servers – this amounts to one failure every 6 days. On our 1 PB GrayWulf server we experienced a much lower disk failure rate ($\sim$1%) so far. These can be dealt with fairly easily, by reformatting simple media errors automatically with data recovery and manually replacing failed disks.

The bigger challenge is that most of the time the storage servers do not need much bandwidth (e.g., during incremental copies), but there is occasionally a need for considerably more bandwidth for a large restore. Our solution is to design the network for the routine scenarios (i.e., incremental backups and small restores). Both the performance servers as well as the storage servers are configured with hotswappable disks so atypical large restores can be performed by physically connecting disks to the servers (i.e., sneakernet [2]). Given that moveable media (disks) are improving faster than networks, sneakernet will inevitably become the low cost solution for large ad hoc restores, e.g., 10–1000 TBs.

The hot-swap disks are also useful for importing and exporting large datasets ($\sim$100 TBs). The Data-Scope is intended to encourage users to visit the facility and bring their own data. For practical reasons, the data set should be small enough to fit in a few 50 pound boxes ($\sim$100 TBs). With the hot swappable feature, users could plug in their disks and have their data copied to the performance servers in a few hours. When visitors leave after a few weeks/months, their data could be swapped out and stored in a bookshelf, where it could be easily swapped back in if the visitor needs to perform a follow-up experiment remotely. Both the performance servers and especially the storage servers could be configured with a few spare disk slots so one can swap in his data without having to swap out someone else's data.

Remote users can transfer data using the fast Open Cloud Consortium (OCC) network [8] – currently a dedicated 10 GbE to MAX and Chicago and soon much higher. OCC has also dedicated network links to several Internet NAPs. Finally, the JHU internal backbone is already running at 10 Gbps and in the next few months the high throughput genomics facilities at JHMI will be connected to this network.

4.4. Usage scenarios

We envisage about 20–25 simultaneous applications, which can use the Data-Scope in four different ways. One can run stable, *high availability public web services*, allowing remote users to perform processing operations on long-lived data sets. These would be typically built on several tens of TB of data and would store data in a redundant fashion for both safety and performance. Examples of such services might be the Virtual Observatory (e.g., usvao.org) cross-match services in astronomy, or the JHU Turbulence database services. Other applications can load their data into a set of *large distributed shared databases*, with aggregate sizes in tens to a few hundred TB. The users can run data intensive batch queries against these data sets and store the intermediate and final results in a shared database and file system space. We have developed a parallel workflow system developed for database ingest of data sets in the 100 TB range for the Pan-STARRS project [10]. This can be turned into a more generic utility with a moderate amount of work. *Hadoop* is an open source implementation of Google's MapReduce [3], which provides a good load balancing and an elegant data-parallel programming paradigm. Part of the instrument will run Hadoop over a multitude of data sets. We will experiment with running the most compute-intensive processing stages (bioinformatics, ray-tracing for image simulations in astronomy) on the GPUs using CUDA code. Finally, when all else is inadequate, certain users can request access to the "*bare metal*", running their own code end-to-end on the performance servers.

4.4.1. *User application toolkits and interfaces*

We will provide users with several general-purpose programming toolkits and libraries to maximize application performance. We have already developed some of the key software components. For example, the custom collaborative environment built for SDSS, Casjobs [5,7], has been in use for seven years, by more than 2500 scientists. The component we need to add is a shared 100 TB intermediate term storage, which has been designed but remains to be built.

We have designed and implemented a generic SQL/CUDA interface that enables users to write their own user defined functions that can execute inside the GPUs, but are called from the database. Since all the data flow is on the backplane of the same server, one can achieve a stunning performance. This was demonstrated in our entry for the SC-09 Data Challenge [16].

We have implemented our own S3/Dropbox lookalike, which has been connected to various open source S3 bindings downloaded from SourceForge. This interface is simple, scalable, well-documented, and will provide a convenient way for users to up- and download their smaller data sets.

On the applications side we have already ported several key applications to CUDA, but most of the development work will materialize as users start to migrate their applications to the Data-Scope. The expectations are that we will need to customize them for the Data-Scope and integrate them to each other. Other components, such as the integration between MPI and the SQL DB, have been prototyped but will need to be fully developed.

In summary, the components that we have developed allow novel high-throughput data analyses. For example, users can, using the SQL-CUDA integration, access powerful analysis patterns like FFT, within a database query. Likewise, Linux MPI applications can read/write data from/to databases using the MPIDB API. During the initial stages of the Data-Scope's development we will use some of the performance and storage servers for software development and testing.

4.4.2. *Data lifecycles*

We envisage three different lifecycle types for data in the instrument. The first would be *persistent data*, over which permanent public services will be built for a wide community, like OpenSkyQuery, the Turbulence database or the Milky Way Laboratory. The main reason to use the Data-Scope in this case is the massive performance gains from the speed of the hardware and parallelism in execution. These data sets will range from several tens to possibly a few hundred TB. The second type of processing will enable truly *massive data processing pipelines* that require both high bandwidth and fast floating point operations. These pipelines will process hundreds of TB, including reprocessing large images from high throughput genomic sequencers, for reduced error rates, and massive image processing tasks for astronomy or cross correlations of large environmental data sets.

Data will be copied physically by attaching 2 TB disks to the Data-Scope, while results will be extracted using the same method. These datasets will be active on the system for one to a few weeks. Another typical user of this pattern would be the LHC data analysis group.

The third type of usage will be *community analysis of very large data sets*. Such datasets will be in the 200–500 TB range. We will keep the media after the dataset has been copied into the instrument and use them to restore the input data in the case of a disk failure. Once such a massive data set arrives, its partitioning and indexing will be massive endeavor therefore it makes only sense if the data stays active for an extended period (3–12 months). Intermediate, derived data sets could also reach tens or even 100 TB. Examples of such datasets include a massive set of simulations (500 cosmological simulations with a high temporal resolution) coupled with an analysis campaign by a broad community.

4.4.3. *System monitoring, data locality*

There will be a data locality server, monitoring the file systems, so the system is aware what's where without depending on a user to update tables manually when disks are swapped in and out. There will be a special file on every disk that tells the system what is on the disk, and a bar code on the outside of disks that would make it easy to locate disks. As for software, we plan to use open source backup software (such as Amanda), as much as possible. Both the operating system software and applications (including Hadoop, SQL Server, etc.) will be deployed using a fully automated environment, already in use for several years at JHU. Since the performance servers are identical, it is very easy to dynamically change the node allocations between the different usage scenarios.

5. 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 tune 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 have also migrated 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. In the near future, we aim to experiment with larger subsets to reflect the recently increased data size limits in SQL Azure (50 GB currently).

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.

To address the analysis needs of the petabyte-scale scientific datasets that are nearly upon us, and recognizing that commercial cloud services do not at present provide a solution to this problem, we are building the Data-Scope – a powerful new instrument designed specifically for this challenge. Intended to be the successor to GrayWulf, Data-Scope will combine vast storage capacity with maximal sequential disk throughput and minimal power requirements. It will achieve these performance goals by choosing, tailoring and optimizing specific technologies best suited for large data movement. To meet competing disk requirements, for instance, the Data-Scope will have a two-tiered architecture consisting of a performance layer and a storage layer. These are the kinds of specializations and customizations required for high-end scientific data analysis that simply are not possible within the commercial cloud services at the moment. The Data-Scope will also accomplish these technical goals while keeping hardware and maintenance costs as low as possible to meet academic budgets.

Acknowledgements

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. The Data-Scope project is supported by an MRI grant from the National Science Foundation (NSF).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia. Above the clouds: a Berkeley view of cloud computing, Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, February 10, 2009.

[2] K. Church and J.A. Hamilton, Sneakernet: clouds with mobility, 2009, available at: http://idies.jhu.edu/meetings/idies09/KChurch.pdf.

[3] J. Dean and S. Ghemawat, MapReduce: simplified data processing on large clusters, in: *6th Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004.

[4] J. Gray, A.S. Szalay, A. Thakar, P. Kunszt, C. Stoughton, D. Slutz and J. van den Berg, Data mining the SDSS SkyServer database, in: *Distributed Data and Structures 4: Records of the 4th International Meeting*, W. Litwin and G. Levy, eds, Paris, France, March 2002, Carleton Scientific, 2003, pp. 189–210, ISBN 1-894145-13-5, also MSR-TR-2002-01, 2002, available at: http://research.microsoft.com/pubs/64558/tr-2002-01.pdf.

[5] N. Li and A.S. Szalay, Simple Provenance in scientific databases, in: *Proc. Microsoft e-Science Workshop*, Pittsburgh, T. Hey and S. Tansley, eds, Microsoft Research, Redmond, WA, USA, 2009, pp. 48–50.

[6] E.H. Neilsen Jr., The Sloan Digital Sky Survey data archive server, *Comput. Sci. Eng.* **10**(1) (2008), 13–17.

[7] W. O'Mullane, N. Li, M.A. Nieto-Santisteban, A. Thakar, A.S. Szalay and J. Gray, Batch is back: CasJobs, serving multi-TB data on the Web, Microsoft Technical Report MSR-TR-2005-19, Microsoft Research, 2005.

[8] Open Cloud Consortium, http://opencloudconsortium.org/.

[9] P. Schmid and A. Ross, Sequential transfer tests of 3.5 hard drives, 2009, available at: http://www.tomshardware.com/reviews/wd-4k-sector,2554-6.html.

[10] Y. Simmhan, R. Barga, C. van Ingen, M. Nieto-Santisteban, L. Dobos, N. Li, M. Shipway, A.S. Szalay, S. Werner and J. Heasley, GrayWulf: scalable software architecture for data intensive computing, in: *Proceedings of the HICSS-42 Conference, Mini-Track on Data Intensive Computing*, I. Gorton, ed., IEEE Computer Society, Washington, DC, USA, 2009, pp. 1–10.

[11] Spider: http://www.nccs.gov/computing-resources/jaguar/file-systems/.

[12] A. Szalay, G. Bell, H. Huang, A. Terzis and A. White, Low-power amdahl-balanced blades for data intensive computing, in: *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09)*, Big Sky, MT, USA, October 10, 2009.

[13] A.S. Szalay, G. Bell, J. van den Berg, A. Wonders, R. Burns, D. Fay, J. Heasley, T. Hey, M. Nieto-SantiSteban, A. Thakar, C. Van Ingen and R. Wilton, GrayWulf: scalable clustered architecture for data intensive computing, in: *Proceedings of the HICSS-42 Conference, Mini-Track on Data Intensive Computing*, I. Gorton, ed., IEEE Computer Society, Washington, DC, USA, 2009, pp. 1–10.

[14] A. Szalay, J. Gray, G. Fekete, P. Kunszt, P. Kukol and A. Thakar, Indexing the sphere with the hierarchical triangular mesh, Microsoft Technical Report, 2005, available at: http://research.microsoft.com/apps/pubs/default.aspx?id=64531.

[15] A.S. Szalay, A.R. Thakar and J. Gray, The SQLLoader data loading pipeline, *Comput. Sci. Eng.* **10**(1) (2008), 38–48.

[16] A.S. Szalay et al., Entry in supercomputing-09 data challenge, in: *SuperComputing 2009 (SC09)*, Portland, OR, USA, November 14–20, 2009.

[17] A.R. Thakar, The sloan digital sky survey: drinking from the fire hose, *Comput. Sci. Eng.* **10**(1) (2008), 9–12.

[18] A.R. Thakar, A.S. Szalay, G. Fekete and J. Gray, The catalog archive server database management system, *Comput. Sci. Eng.* **10**(1) (2008), 30–37.