GrayWulf: Scalable Clustered Architecture for Data Intensive Computing

Alexander S. Szalay\textsuperscript{1}, Gordon Bell\textsuperscript{2}, Jan Vandenberg\textsuperscript{1}, Alainna Wonders\textsuperscript{1}, Randal Burns\textsuperscript{1}, Dan Fay\textsuperscript{2}, Jim Heasley\textsuperscript{3}, Tony Hey\textsuperscript{2}, Maria Nieto-Santisteban\textsuperscript{1}, Ani Thakar\textsuperscript{1}, Catherine Van Ingen\textsuperscript{2}, Richard Wilton\textsuperscript{1}

1. The Johns Hopkins University, 2. Microsoft Research, 3. The University of Hawaii

\texttt{szalay@jhu.edu, gbell@microsoft.com, jvv@jhu.edu, alainna@pha.jhu.edu, randal@cs.jhu.edu, dan.fay@microsoft.com, heasley@ifa.hawaii.edu, tony.hey@microsoft.com, nieto@pha.jhu.edu, thakar@jhu.edu, vaningen@windows.microsoft.com, rwilton@pha.jhu.edu}

Abstract

Data intensive computing presents a significant challenge for traditional supercomputing architectures that maximize FLOPS since CPU speed has surpassed IO capabilities of HPC systems and BeoWulf clusters. We present the architecture for a three tier commodity component cluster designed for a range of data intensive computations operating on petascale data sets. The design goal is a balanced system in terms of IO performance and memory size, according to Amdahl’s Laws. GrayWulf pays tribute to Jim Gray who stimulated the system and its design. The hardware currently installed at JHU exceeds one petabyte of storage and has 0.5 bytes/sec of I/O and 1 byte of memory for each CPU cycle. The GrayWulf provides almost an order of magnitude better balance than existing systems. The paper covers its architecture and reference applications. The software design is presented in a companion paper.

1. Trends of Scientific Computing

The nature of high performance computing is changing. While a few years ago much of high-end computing involved maximizing CPU cycles per second allocated for a given problem; today it revolves around performing computations over large data sets. This means that efficient data access from disks and data movement across servers is an essential part of the computation.

Data sets are doubling every year, growing slightly faster than Moore’s Law\cite{1}. This is not an accident. It reflects the fact that scientists are spending an approximately constant budget on more capable computational facilities and disks whose sizes have doubled annually for over a decade. The doubling of storage and associated data is changing the scientific process itself, leading to the emergence of eScience – as stated by Gray’s Fourth Paradigm of Science based on Data Analytics\cite{2}.

Much data is observational, due to the rapid emergence of successive generations of inexpensive electronic sensors. At the same time large numerical simulations are also generating data sets with increasing resolutions, both in the spatial and temporal sense. These data sets are typically tens to hundreds of terabytes\cite{3,4}. As a result, scientists are in a dire need of a scalable solution for data-intensive computing.

The broader scientific community has traditionally preferred to use inexpensive computers to solve their computational problems, rather than remotely located high-end supercomputers. First they used VAXes in the 80s followed by low-cost workstations. About 10 years ago it became clear that the computational needs of many scientists exceeded that of a single workstation and many users wanted to avoid the large, centralized supercomputer centers. This was when laboratories started to build computational clusters from commodity components. The idea and phenomenal success of the BeoWulf cluster\cite{5} shows that scientists (i) prefer to have a solution that is under their direct control, (ii) are quite willing to use existing proven and successful templates, and, (iii) generally want a ‘do-it-yourself’ inexpensive solution.

As an alternative to ‘building your own cluster’, bringing computations to the free computer resource became a successful paradigm, Grid Computing\cite{6}. This self-organizing model, where groups of scientists pool computing resources irrespective of their physical location suits applications that require lots of CPU time with relatively little data movement.

For data intensive applications the concept of ‘cloud computing’ is emerging where data and computing are co-located at a large centralized facility, and accessed as well-defined services. This offers many advantages over the grid-based model
and is especially applicable where many users access shared common datasets. It is still not clear how willing scientists will be to use such remote clouds[7]. Recently Google and IBM have made such a facility available for the academic community.

Due to these data intensive scientific problems a new challenge is emerging, as many groups in science (but also beyond) are facing analyses of data sets in tens of terabytes, eventually extending to a petabyte since disk access and data-rates have not grown with their size. There is no magic way to manage and analyze such data sets today. The problem exists both on the hardware and the software levels.

The requirements for the data analysis environment are (i) scalability, including the ability to evolve over a long period, (ii) performance, (iii) ease of use, (iv) some fault tolerance and (v) most important—low entry cost.

2. Database-Centric Computing

2.1. Bring analysis to the data, not vice-versa

Many of the typical data access patterns in science require a first, rapid pass through the data, with relatively few CPU cycles carried out on each byte. These involve filtering by a simple search pattern, or computing a statistical aggregate, very much in the spirit of a simple mapping step of MapReduce[8]. Such operations are also quite naturally performed within a relational database, and expressed in SQL. So a traditional relational database fits these patterns extremely well.

The picture gets a little more complicated when one needs to run a more complex algorithm on the data, not necessarily easily expressed in a declarative language. Examples of such applications can include complex geospatial queries, processing time series data, or running the BLAST algorithm for gene sequence matching.

The traditional approach of bringing the data to where there is an analysis facility is inherently not scalable, once the data sizes exceed a terabyte due to network bandwidth, latency, and cost. It has been suggested[2] that the best approach is to bring the analysis to the data. If the data are stored in a relational database, nothing is closer to the data than the CPU of the database server. It is quite easy today with most relational database systems to import procedural (even object oriented) code and expose their methods as user defined functions within the query.

This approach has proved to be very successful in many of our reference applications, and while writing class libraries linked against SQL was not always the easiest coding paradigm, its excellent performance made the coding effort worthwhile.

2.2. Typical scientific workloads

Over the last few years we have implemented several eScience applications, in experimental data-intensive physical sciences applications such as astronomy, oceanography and water resources. We have been monitoring the usage and the typical workloads corresponding to different types of users. When analyzing the workload on the publicly available multi-terabyte Sloan Digital Survey SkyServer database[9], it was found that most user metrics have a $1/f$ power law distribution[10].

Of the several hundred million data accesses most queries were very simple, single row lookups in the data set, which heavily used indices such as on position over the celestial sphere (nearest object queries). These made up the high frequency, low volume part of the power law distribution. On the other end there were analyses which did not map very well on any of the precomputed indices, thus the system had to perform a sequential scan, often combined with a merge join. These often took over an hour to scan through the multi-terabyte database. In order to submit a long query, users had to register with an email address, while the short accesses were anonymous.

2.3. Advanced user patterns

We have noticed a pattern in-between these two types of accesses. Long, sequential accesses to the data were broken up into small, templated queries, typically implemented by a simple client-side Python script, submitted once in every 10 seconds. These “crawlers” had the advantage from the user’s perspective of returning data quickly, and in small buckets. If the inspection of the first few buckets hinted at an incorrect request (in the science sense), the users could terminate the queries without having to wait too long.

The “power users” have adopted a different pattern. Their analyses typically involve a complex, multi-step workflow, where the correct end result is approached in a multi-step hit-and-miss fashion. Once they zoom in on a final workflow, they execute it over the whole data set, by submitting a large job into a batch queue. In order to support this, we have built “MyDB”, a server-side workbench environment[11], where users
get their own database with enough disk space to store all the intermediate results. Since this is server-side, the bandwidth is very high, even though the user databases reside on a separate server. Users have full control over their own databases, and they are able to perform SQL joins with all the data tables in the main archive.

The workbench also supports easy upload of user data into the system, and a collaborative environment, where users can share tables with one another. This environment has proved itself to be incredibly successful. Today, 1,600 astronomers, approximately 10 percent of the world’s professional astronomy population, are daily users of this facility.

In summary, most scientific analyses are done in an exploratory fashion, where “everything goes”, and few predefined patterns apply. Users typically want to experiment, try many innovative things that often do not fit preconceived notions, and would like to get very rapid feedback on the momentary approach. In the next sections we will discuss how we expand this framework and environment substantially beyond the terabyte scale of today.

3. Building Balanced Systems

3.1. Amdahl’s laws

Amdahl has established several laws for building a balanced computer system [12]. These were reviewed recently[13] in the context of the explosion of data. The paper pointed out that contemporary computer systems IO subsystems are lagging CPU cycles. In the discussion below we will be concerned with two of Amdahl’s Laws:

A balanced system

- needs one bit of IO for each CPU cycle
- has 1 byte of memory for each CPU cycle

These laws enumerate a rather obvious statement— in order to perform continued generic computations, we need to be able to deliver data to the CPU, through the memory. Amdahl observed that these ratios need to be close to unity and this need has stayed relatively constant.

The emergence of multi-level caching led to several papers pointing out that a much lower IO to MIPS ratio coupled with a large enough memory can still provide a satisfactory performance[14]. While this is true for problems that mostly fit in memory, it fails to extend to computations that need to process so much data (PB) that they must reside on external disk storage. At that point having a fast memory cache is not much help, since the bottleneck is disk IO.

3.2. Raw sequential IO

For very large data sets the only way we can even hope to accomplish the analysis if we follow a maximally sequential read pattern. Over the last 10 years while disk sizes have increased by a factor of 1,000, the rotation speed of large disks used in disk arrays has only changed a factor of 2 from 5,400 rpm to 10,000 rpm. Thus random access times of disks have only improved about 7% per year.

The sequential IO rate has grown somewhat faster as the density of the disks has increased by the square root of disk capacity. For commodity SATA drives the sequential IO performance is typically 60MB/sec, compared with 20MB/sec 10 years ago. Nevertheless, compared to the increase of the data volumes and the CPU speedups, this increase is not fast enough to conduct business as usual. Just loading a terabyte at this rate takes 4.5 hours. Given this sequential bottleneck, the only way to increase the disk throughput of the system is to add more and more disk drives and to eliminate obvious bottlenecks in the rest of the system.

3.3. Scale-up or scale-out?

A 20-30TB data set is too large to fit on a single, inexpensive server. One can scale-up, buying an expensive multiprocessor box with many fiber channel (FC) Host Channel Adapters (HCA) and a FC disk array, easily exceeding the $1M price tag. The performance of such systems is still low, especially for sequential IO. To build a system with over one GB/sec sequential IO speed one needs at least 8 FC adapters. While this may be attractive for management, the entry cost is not low!

Scaling out using a cluster of disks attached to each computing node provides a much more cost effective and high throughput solution, very much along the lines of BeoWulf designs. The sequential read speed of a properly balanced mid-range server with many local disks can easily exceed a GB/sec before saturation[15]. The cost of such a server can be kept close to the $10,000 range. On the other hand managing an array of such systems, and manually partitioning the data can be quite a challenge. Instead of mid-range servers the scale-out can be done on lower-end machines, and deployed to a very large number (~100,000), as done by Google.

Given the success of the BeoWulf concept for academic research, we believe that the dominant
solution in this environment will be deployed locally. Given the scarcity of space at universities it also needs to have a high packing density.

4. The GrayWulf System

4.1. Overall Design Principles

We are building a combined hardware and software platform from commodity components to perform large-scale database-centric computations. The system should

a) scale to petabyte-size data sets
b) provide very high sequential bandwidth to data
c) support most eScience access patterns
d) provide simple tools for database design
e) provide tools for fast data ingest

This paper describes the system hardware and the hardware monitor tools. A second paper describes the software tools that provide functionality for (c) – (e).

4.2. Modular, layered architecture

Our cluster consists of modular building blocks, in three tiers. Having multiple tiers provides a system with a certain amount of hierarchical spread of memory and disk storage. The low level data can be spread evenly among server nodes on the lowest tier, all running in parallel, while query aggregations are done on more powerful servers in the higher tiers.

The lowest, tier 1 building block is a single 2U sized Dell 2950 server, with two quad core 2.66GHz CPUs. Each server has 16 GB of memory, two PCIe PERC6/E dual-channel RAID controllers and a 20 Gbit/sec SilverStorm Infiniband HCA, with a PCIe interface. Each server is connected to two 3-U MD1000 SAS disk boxes that contain a total of 30-750 GB, 7,200 rpm SATA disks. Each disk box is connected to its dedicated dual-channel controller (see section 4.3). There are two mirrored 73 GB, 15,000 rpm disks residing in internal bays, connected to a controller on the motherboard. These disks contain the operating system and the rest of the installed software. Thus, each of these modules takes up 8 rack units, and contains a total of 22.5TB of data storage. Four of these units with UPS power is put in a rack. The whole lower tier consists of 10 such racks, with a total of 900TB of data space, and 640 GB of memory.

Tier 2 consists of four Dell R900 servers with 16 cores each and 64 GB of memory, connected to three of the MD1000 disk boxes, each populated as above. There is one dual channel PERC6/E controller for each disk box. The system disks are two mirrored 73GB SAS drives at 10,000 rpm and a 20Gbit/sec SilverStorm Infiniband HCA. This layer has a total of 135TB of data storage and 256GB of memory. We also expect that data sets that need to be sorted and/or rearranged will be moved to these servers, utilizing the larger memory.

Finally, tier 3 consists of two Dell R900 servers with 16 cores, 128 GB of memory, each connected to a single MD1000 disk box with 15 disks, and a SilverStorm IB card. The total storage is 22.5TB and the memory is 256 GB. These servers can also run some of the resource intensive applications, complex data intensive web services (still inside the SQL Server engine using CLR integration) which require more physical memory than available on the lower tiers.

<table>
<thead>
<tr>
<th>Tier</th>
<th>server</th>
<th>core</th>
<th>mem[GB]</th>
<th>disk[TB]</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tier 1</td>
<td>2950</td>
<td>8</td>
<td>16</td>
<td>22.50</td>
<td>40</td>
</tr>
<tr>
<td>Tier 2</td>
<td>R900</td>
<td>16</td>
<td>64</td>
<td>33.75</td>
<td>4</td>
</tr>
<tr>
<td>Tier 3</td>
<td>R900</td>
<td>16</td>
<td>128</td>
<td>11.25</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
<td>416</td>
<td>1152</td>
<td>46</td>
</tr>
</tbody>
</table>

Table 1. Tabular description of the three tiers of the cluster with aggregates for cores, memory and disk space within our GrayWulf system.

The Infiniband interconnect is through a Qlogic SilverStorm 9240 288 port switch, with across-sectional aggregate bandwidth of 11.52 Tbit/s. The switch also contains a 10 Gbit/sec Ethernet module that connects any server to our dedicated single lambda National Lambda Rail connection over the Infiniband fabric, without the need for dedicated 10 Gbit Ethernet adapters for the servers.

Initial Infiniband testing suggests that we should be able to utilize at least the Infiniband Sockets Direct Protocol[16] for communication between SQL Server instances, and that the SDP links should sustain at least 800-850 MB/sec. Of course, we hope to achieve the ideal near-wirespeed throughput of the 20 Gbit/sec fabric. This seems feasible, as we will have ample opportunity to tune the interconnect, and the Windows Infiniband stack itself is evolving rapidly these days.
The cluster is running Windows Enterprise Server 2008 and the database engine is SQL Server 2008 that is automatically deployed across the cluster.

4.3. Balanced IO bandwidth

The most important consideration when we designed the system (besides staying within our budget) was to avoid the obvious choke points in terms of streaming data from disk to CPU, then across the interconnect layer. These bottlenecks can exist all over the system: the storage bus (FC, SATA, SAS, SCSI), the storage controllers, the PCI buses, system memory itself, and in the way that software chooses to access the storage. It can be tricky to create a system that dodges all of them.

The disks: A single 7,200 rpm 750 GB SATA drive can sustain about 75 MB/sec sequential reads at the outer cylinders, and somewhat less on the inner parts.

The storage interconnect: We are using Serial Attached SCSI (SAS) to connect our SATA drives to our systems. SAS is built on full-duplex 3 Gbit/sec “lanes”, which can be either point-to-point (i.e. dedicated to a single drive), or can be shared by multiple drives via SAS “expanders”, which behave much like network switches. Prior parallel SCSI standards like Ultra320 accommodated only expensive native SCSI drives, which are great for IOPS-driven applications, but are not as compelling for petascale, sequentially-accessed data sets. In addition to supporting native SAS/SCSI devices, SAS also supports SATA drives, by adopting a physical layer compatible with SATA, and by including a Serial ATA Tunneling Protocol within the SAS protocol. For large, fast, potentially low-budget storage applications, SATA over SAS is a terrific compromise between enterprise-class FC and SCSI storage, and the inexpensive but fragile “SATA bricks” which are particularly ubiquitous in research circles.

The disk enclosures: Each Dell MD1000 15-disk enclosure uses a single SAS “4x” connection. 4x is a bundle of four 3 Gbit/sec lanes, carried externally.

The SCSI protocol itself operates with a 25% bus overhead. So for a 3 Gbit/sec SAS lane, the real-world sustainable throughput is about 225 MB/sec. The Serial ATA Tunneling Protocol introduces an additional 20% overhead, so the real-world SAS-lane throughput is about 180 MB/s when using SATA drives.

The disk enclosures: Each Dell MD1000 15-disk enclosure uses a single SAS “4x” connection. 4x is a bundle of four 3 Gbit/sec lanes, carried externally.
over a standard Infiniband-like cable with Infiniband-like connectors. This 12 Gbit/sec connection to the controller is very nice relative to common 4 Gbit/sec FC interconnects. But with SATA drives, the actual sustainable throughput over this 12 Gbit/sec connection is 720 MB/sec. Thus we have already introduced a moderate bottleneck relative to the ideal ~1100 MB/sec throughput of our 15 750 MB/sec drives. For throughput purposes, only about 10 drives are needed to saturate an MD1000 enclosure’s SAS backplane.

The disk controllers: The LSI Logic based Dell PERC6/E controller has dual 4x SAS channels, and has a feature set which is common among contemporary RAID controllers. Why do we go to the trouble and the expense of using one controller per disk enclosure when we could easily attach one dedicated 4x channel to each enclosure using a single controller? Our tests show that the PERC6 controllers themselves saturate at about 800 MB/sec, so to gain additional throughput as we add more drives, we need to add more controllers. It is convenient that a single controller is so closely matched to a SATA-populated enclosure.

The PCI and memory busses: The Dell 2950 servers have two “x8” PCI Express connections and one “x4” connection, rated at 2000 MB/sec and 1000 MB/s half-duplex speeds respectively. We can safely use the x4 connection for one of our PERC6 controllers since we expect no more than 720 MB/s from these. The 2000 MB/sec-each x8 connections are plenty for one of the PERC6 controllers, and just enough for our 20 Gbit/sec DDR Infiniband HCAs. Our basic tests suggest that the 2950 servers can read from memory at 5700 MB/sec, write at 4100 MB/sec, and copy at 2300 MB/sec. This is a pretty good match to our 1440 MB/sec of disk bandwidth and 2000 MB/sec Infiniband bandwidth, though in the ideal case with every component performing flat-out, the system backplane itself could potentially slow us down a bit.

Test methodology: We use a combination of Jim Gray’s MemSpeed tool, and SQLIO [17]. MemSpeed measures system memory performance itself, along with basic buffered and unbuffered sequential disk performance. SQLIO can perform various IO performance tests using IO operations that resemble what SQL Server’s. Using SQLIO, we typically test sequential reads and writes, and random IOPS, but we’re most concerned with sequential read performance. Performance measurements presented here are typically based on SQLIO’s sequential read test, using 128 KB requests, one thread per system processor, and 32-deep requests per thread. We believe that this resembles the typical table scan behavior of SQL Server’s Enterprise Edition. We find that the IO speeds that we measure with SQLIO are very good predictors for SQL Server’s real-world IO performance.

In Figure 3, we present our measurements of the saturation points of various components of the GrayWulf’s IO system. The labels on the plots designate the number of controllers, the number of disk boxes, and the number of SAS lanes for each experiment. The “1C-1B-2S” plot shows a pair of 3 Gbit/sec SAS lanes saturating near the expected 360
MB/sec mark. “1C-1B-4S” shows the full “4x” SAS connection of one of the MD1000 disk boxes saturating at the expected 720 MB/sec. “1C-2B-8S” demonstrates that the PERC6 controller saturates at just under 1 GB/sec. “2C-2B-8S” shows the performance of the actual Tier 1 GrayWulf nodes, right at twice the “1C-1B-4S” performance.

The full cluster contains 96 of the 720 MB/sec PERC6/MD1000 building blocks. This translates to an aggregate low-level throughput of about 70 GB/sec. Even though the bandwidth of the interconnect is slightly below that of the disk subsystem, we do not regard this as a major bottleneck, since in our typical applications the data is first filtered and/or aggregated, before it is sent across the network for further stream aggregation. This filtering operation will result in a reduction of the data volume to be sent across the network (for most scenarios) thus a factor of 2 lower network throughput compared to the disk IO is quite tolerable.

The other factor to note is that for our science application the relevant calculations take place at the backplanes of the individual servers, and the higher level aggregation requires a much lower bandwidth at the upper tiers.

4.4. Monitoring Tools

The full-scale GrayWulf system is rather complex, with many components performing tasks in parallel. We need a detailed performance monitoring subsystem that can track and quantitatively measure the behavior of the hardware. We need the performance data in several different contexts:

- track and monitor the status of computer and network hardware in the “traditional” sense
- as a tool to help design and tune individual SQL queries, monitor level of parallelism
- track the status of long-running queries, particularly those that are heavy consumers of CPU, disk, or network resources in one or more of the GrayWulf machines

The performance data are acquired both from the well-known “PerfMon” (Windows Performance Data Helper) counters and from selected SQL Server Dynamic Management Views (DMVs). To understand the resource utilization of different long-running GrayWulf queries, it is useful to be able to relate DMV performance observations of SQL Server objects such as filegroups with PerfMon observations of per-processor CPU utilization and logical disk volume IO.

Performance data for SQL queries are gathered by a C# program that monitors SQL Trace events and samples performance counters on one or more SQL Servers. The data is aggregated in a SQL database, where performance data is associated with individual SQL queries. This part of the monitoring represented a particular challenge in a parallel environment, since SQL Server does not provide an easy mechanism to follow process identifiers for remote subqueries. Data gathering is limited to “interesting” SQL queries, which are annotated by specially-formatted SQL comments whose contents are also recorded in the database.

5. Reference Applications

We have several reference applications, each corresponding to a different kind of data layout, and thus a different access pattern. These range from computational fluid dynamics to astronomy, each consisting of datasets close to or exceeding 100TB.

5.1. Immersive Turbulence

The first application is in computational fluid dynamic, CFD, to analyze large hydrodynamic simulations of turbulent flow. The state-of-the-art simulations have spatial resolutions of 4096 and consist of hundreds if not thousands of timesteps. While current supercomputers can easily run these simulations it is becoming increasingly difficult to perform subsequent analyses of the results. Each timestep over such a spatial resolution can be close to a terabyte. Storing the data from all timesteps requires a storage facility reaching hundreds of terabytes. Any analysis of the data requires the users to analyze these data sets, which requires accessing the same compute/storage facility. As the cutting edge simulations become ever larger, fewer and fewer scientists can participate in the subsequent analysis. A new paradigm is needed, where a much broader class of users can perform analyses of such data sets.

A typical scenario is that scientists want to inject a number of particles (5,000-50,000) into the simulation and follow their trajectories. Since many of the CFD simulations are performed in Fourier space, over a regular grid, no labeled particles exist in the output data. At JHU we have developed a new paradigm to interact with such data sets using a web-services interface [18].

A large number of timesteps are stored in the database, organized along a convenient three-dimensional spatial index based on a space-filling curve (Peano-Hilbert, or z-transform). The disk layout closely preserves the spatial proximity of grid
cells, making disk access of a coherent region more sequential. The data for each timestep is simply sliced across N servers, shown as scenario (a) on Figure 4. The slicing is done along a partitioning key derived from the space filling curve.

Spatial and temporal interpolation functions are implemented inside the database that can compute the velocity field at an arbitrary spatial and temporal coordinate. A scientist with a laptop can insert thousands of particles into the simulation by requesting the velocity field at those locations. Given the velocity values, the laptop can then integrate the particles forward, and again request the velocities at the updated location and so on. The resulting trajectories of the particles have been integrated on the laptop, but they correspond to the velocity field inside the simulation spanning hundreds of terabytes. This is digital equivalent of launching sensors into a vortex of a tornado, like the scientists in the movie “Twister”.

This computing model has been proven extremely successful; we have so far ingested a $1024^3$ simulation into a prototype SQL Server cluster, and created the above mentioned interpolating functions configured as a TVF (table valued function) inside the database[19]. The data has been made publicly available. We also created a Fortran(!) harness to call the web service, since most of the CFD community is still using that language.

5.2. SkyQuery

The SkyQuery[20] service has been originally created as part of the National Virtual Observatory. It is a universal web services based federation tool, performing cross-matches (fuzzy geospatial joins) over large astronomy data sets. It has been very successful, but has a major limitation. It is very good in handling small areas of the sky, or small user-defined data sets. But as soon as a user requests a cross-match over the whole sky, involving the largest data sets, generating hundreds of millions of rows, its efficiency rapidly deteriorates, due to the slow wide area connections.

Co-locating the data from the largest few sky surveys on the same server farm will give a dramatic performance improvement. In this case the cross-match queries are running on the backplane of the database. We have created a zone-based parallel algorithm that can perform such spatial cross-matches in the database[21] extremely fast. This algorithm has also been shown to run efficiently over a cluster of databases. We can perform a match between two datasets (2MASS, 400M objects and USNOB, 1B objects) in less than 2 hours on a single server. Our reference application for the GrayWulf is running parallel queries, and merging the result set, using a paradigm similar to the MapReduce algorithm[8].

Making use of threads and multiple servers we believe that on the JHU cluster can achieve a 20-fold speedup, yielding a result in a few minutes instead of a few hours. We use our spatial algorithms to compute the common sky area of the intersecting survey footprints then split this area equally among the participating servers, and include this additional spatial clause in each instance of the parallel queries for an optimal load balancing. The data layout in this case is a simple N-way replication of the data, as shown as part (b) on Figure 4. The relevant database that contains all the catalogs is about 5TB, thus a 20-way replication is still manageable. The different query streams will be aggregated on one of the Tier 3 nodes.

5.3. Pan-STARRS

The Pan-STARRS project[4] is a large astronomical survey, that will use a special telescope in Hawaii with a 1.4 gigapixel camera to sample the sky over a period of 4 years. The large field of view and the relatively short exposures will enable the telescope to cover three quarters of the sky 4 times per year, in 5 optical colors. This will result in more than a petabyte of images per year. The images will then be processed through an image segmentation pipeline that will identify individual detections, at the rate of 100 million detections per night. These detections will be associated with physical objects on
the sky and loaded into the project’s database for further analysis and processing. The database will contain over 5 billion objects and well over 100 billion detections. The projected size of the database is 30 terabytes by the end of the first year, growing to 80 terabytes by the end of year 4.

Expecting that most of the user queries will be run against the physical object, it is natural to consider a hierarchical data layout, shown of section (c) on Figure 4. The star schema of the database naturally provides a framework for such an organization. The top level of the hierarchy contains the objects, which are logically partitioned into $N$ segments, but they physically stored on one of the Tier 2 servers. The corresponding detections (much larger in cardinality) are then sliced among the $N$ servers in the lowest Tier ($A'$,$B'$, etc).

### 6. Comparisons to Other Architectures

In this section we would like to consider several well-studied architectures for scientific High Performance Computing and calculate their Amdahl numbers for comparison. The *Amdahl RAM number* is calculated by dividing the total memory in Gbytes with the aggregate instruction cycles in units of GIPS (1000 MIPS). The Amdahl IO number is computed by dividing the aggregate sequential IO speed of the system in Gbits/sec by the GIPS value. A ratio close to 1 indicates a balanced system in the Amdahl sense.

We consider first a typical University Beowulf cluster, consisting of 50 3GHz dual-core machines, each with 4GB of memory and one SATA disk drive with 60MB/sec. Next, we consider a typical desktop used by the average scientist, doing his/her own data analysis. Today such a machine has 2 3GHz CPUs, 4GB of memory and 3 SATA disk drives, which provide an aggregate sequential IO of about 150MB/sec, since they all run off the motherboard controller. A virtual machine in a commercial cloud would have a single CPU, say at 3GHz, 4GB RAM, but a lower IO speed of about 30MB/sec per VM instance[7].

Let us consider two hypothetical machines used in today’s scientific supercomputing environments. An approximate configuration “SC1” for a typical BlueGene-like machine was obtained from the LLNL web pages[22]. The sequential IO performance of an IO-optimized BlueGene/L configuration with 256 IO nodes has been measured to reach 2.6 GB/sec peak[23]. A simple minded scaling this result to the 1664 IO nodes in the LLNL system gives us the hypothetical 16.9 GB/sec figure used in the table for “SC1”. The other hypothetical supercomputer, “SC2,” has been modeled on the Cray XT-3 at the Pittsburgh Supercomputer Center. The XT-3 IO bandwidth is currently limited by the PSC Infiniband fabric[24]. We have also attempted to get accurate numbers from several of the large cloud computing companies – our efforts have not been successful, unfortunately.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Beowulf</td>
<td>100</td>
<td>300</td>
<td>200</td>
<td>3000</td>
<td>0.67</td>
<td>0.080</td>
</tr>
<tr>
<td>Desktop</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>150</td>
<td>0.67</td>
<td>0.200</td>
</tr>
<tr>
<td>Cloud VM</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>30</td>
<td>1.33</td>
<td>0.080</td>
</tr>
<tr>
<td>SC1</td>
<td>212992</td>
<td>150000</td>
<td>18600</td>
<td>16900</td>
<td>0.12</td>
<td>0.001</td>
</tr>
<tr>
<td>SC2</td>
<td>2090</td>
<td>5000</td>
<td>8260</td>
<td>4700</td>
<td>1.65</td>
<td>0.008</td>
</tr>
<tr>
<td>GrayWulf</td>
<td>416</td>
<td>1107</td>
<td>1152</td>
<td>70000</td>
<td>1.04</td>
<td>0.506</td>
</tr>
</tbody>
</table>

Table 2. The two Amdahl numbers characterizing a balanced system are shown for a variety of systems commonly used in scientific computing today. Amdahl numbers close to 1 indicate a balanced architecture.

7. Summary

The Graywulf IO numbers have been estimated from our single-node measurements of sequential IO performance and our typical reference workloads. Table 2 shows that our GrayWulf architecture excels in aggregate IO performance as well as in the Amdahl IO metric, in some cases well over a factor of 50. It is interesting that the desktop of a data intensive user comes closest to the GrayWulf IO number of 0.5.
In this paper we wanted to make a few simple points:

- Data-intensive scientific computations today require a large sequential IO speed more than anything else.
- As we consider higher and higher end systems, their IO rate does not keep up with the CPUs.
- It is possible to build balanced IO intensive systems using commodity components
- The total cost of the system (excluding the Infiniband) is well under $800K
- Our system satisfies the criteria in today’s data-intensive environment similar to those that made the original BeoWulf idea so successful.

8. Acknowledgements

The authors would like to thank Jim Gray for many years of intense collaboration and friendship. Financial support for the GrayWulf cluster hardware was provided by the Gordon and Betty Moore Foundation, Microsoft Research and the Pan-STARRS project. Microsoft’s SQL Server group, in particular Lubor Kollar and Jose Blakeley has given us enormous help in optimizing the throughput of the database engine.

9. References

10.
[22]https://computing.llnl.gov/?set=resources&page=SCF_resources#bluegenel
[24] Ralph Roskies, private communication