Using the Active Storage Fabrics Model to Address Petascale Storage Challenges

Blake G. Fitch
IBM T.J. Watson Research Center
1101 Kitchawan Road/Route 134
Yorktown Heights, NY 10598
bgf@us.ibm.com

Aleksandr Rayshubskiy
IBM T.J. Watson Research Center
1101 Kitchawan Road/Route 134
Yorktown Heights, NY 10598
arayshu@us.ibm.com

Michael C. Pitman
IBM T.J. Watson Research Center
1101 Kitchawan Road/Route 134
Yorktown Heights, NY 10598
pitman@us.ibm.com

T.J. Christopher Ward
IBM Software Group
Hursley Park
Hursley SO212JN, United Kingdom
tjcw@uk.ibm.com

Robert S. Germain
IBM T.J. Watson Research Center
1101 Kitchawan Road/Route 134
Yorktown Heights, NY 10598
rgermain@us.ibm.com

ABSTRACT

We present the Active Storage Fabrics (ASF) model for storage embedded parallel processing as a way to address petascale data intensive challenges. ASF is aimed at emerging scalable system-on-a-chip, storage class memory architectures, but may be realized in prototype form on current parallel systems. ASF can be used to transparently accelerate host workloads by close integration at the middleware data/storage boundary or directly by data intensive applications. We provide an overview of the major components involved in accelerating a parallel file system and a relational database management system, describe some early results, and outline our current research directions.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management; D.4.2 [**Operating Systems**]: Storage Management—*main memory, distributed memories*

General Terms

Storage, Parallel

Keywords

Active storage, in-memory database, data-intensive computing

Supercomputing PDSW '09, Nov. 15, 2009. Portland, OR, USA

1. INTRODUCTION

We describe the Active Storage Fabrics (ASF) model which addresses several petascale data intensive computing challenges. Applications exploiting petascale computational platforms can create datasets that are effectively too large to move and consequently require analyses to be conducted in place within the system where they are stored. As scientific communities reach consensus on computational methods and tools, we can expect increased reuse of large datasets produced by either simulation or experiment, effectively driving the creation of scientific data warehouses. These warehouses will require standardized access to raw data with efficient means of custom data reduction, analysis, and presentation.

Consider the case of an investigator wishing to conduct a variety of ad-hoc, possibly computationally intensive analyses on a massive dataset. Current practice for analysis of massive datasets often involves many trips across the IO bottleneck between the parallel processor that generates or analyzes the data and the secondary storage system. The classic HPC model of a massively parallel machine connected through an industry standard network (Ethernet or InfiniBand) to a distributed file system will be strained by such a use case because of the increasing relative cost of an IO system that matches the data production capabilities of a petascale data center.

The Active Storage Fabrics model is informed by these emerging data and computationally intensive use cases and recognizes a need to preserve standard user and middleware interfaces. The use of standard interfaces preserves the value of the current user skill-set and also helps maintain interoperability with existing data life cycle management tools. The Active Storage Fabrics

concept focuses on embedding parallel computation within distributed data, enabling parallel and/or serial job steps to interoperate through common data access methods, and the modification of middleware at the data/storage interface in order to transparently exploit those two capabilities. Overall, scientific data warehousing with the characteristic of repeated analyses on large, immobile, and shared datasets, along with technology trends such as the end of processor frequency scaling, the cost of I/O bandwidth, and emergence of storage class memory (SCM) technology[5], demands a new approach to the relationship between computation and storage which is currently being explored[2, 1].

The target hardware architecture for the Active Storage Fabrics work can be described as a large number of Scalable, System-on-a-chip, Storage class Memory nodes, referred to here as "S3M", connected to personal computers, large servers, or massively parallel HPC systems. The S3M nodes provide a medium for active storage - a current Blue Gene machine with Flash memory at each node makes a good conceptual model. Early micro-benchmarks shown in Figure 1 support the feasibility of this model and of evolving the Blue Gene architecture into a full S3M platform. The ASF client can be embedded in middleware on a standard server or in HPC jobs so they can drive data directly into the active storage. Currently, true S3M nodes with substantial amounts of persistent memory are too costly to build at scale. Development of system software and applications for this hardware model on currently available systems should position ASF to fully utilize cost effective persistent memories as they become available.

ASF provides a framework for allowing applications and middleware to offload computation into storage where the "storage" is actually backed by the memory (DRAM and/or SCM) of a parallel machine. Effectively, the aggregate memory of a large parallel computer presents storage class memory characteristics without persistence. The capability of Blue Gene class machines to be data intensive computational platforms enabled early ASF development but the architectural target is defined by the promise of economical S3M platforms. We have focused on integrating ASF with major middleware packages, namely the IBM General Parallel File System (GPFS) and IBM DB2 by prototyping ASF using IBM Blue Gene/L and Blue Gene/P systems. Integrated in this way, ASF enables middleware packages to transparently accelerate selected modules while maintaining legacy user interfaces of those packages.

We present an overview of the Active Storage Fabrics model, describe two middleware integration exercises that exploit ASF prototypes to enable transparent acceleration, and show some early performance results. We outline current development objectives and future research directions.

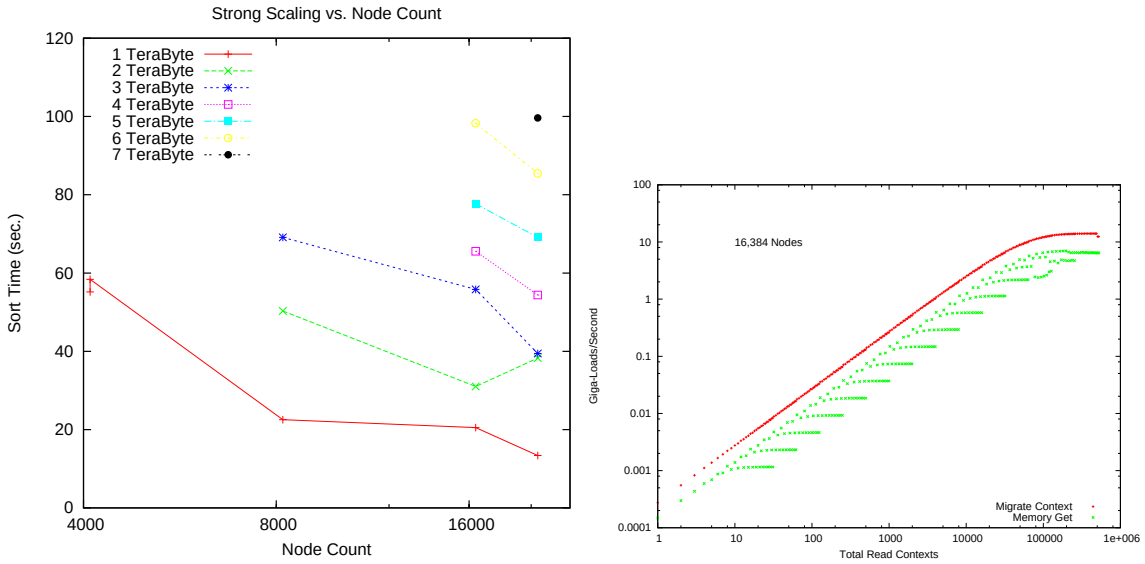
2. ACTIVE STORAGE FABRICS OVERVIEW

ASF is a collection of components that surround a parallel in-memory database (PIMD). PIMD is a parallel client, parallel server, key/value object store. While the specific design of PIMD is shaped by the S3Mtarget platform, such hash-based key/value containers are used extensively for search, file-sharing, and other applications[4, 3]. PIMD stores key/value records in containers called Partitioned Data Sets (PDS) which are distributed in the parallel server. PIMD clients may run on the same nodes as the server or may be external to the Active Storage Fabric. PIMD can respect Unix style access permissions on a per PDS basis (although this is not currently enforced). PIMD hashes records into the fabric and maintains load balance by data redistribution.

Maintaining balance in the data distribution is important as the node count increases because each node contains a proportionally smaller fraction of the total capacity. The PIMD server is allowed to relocate records if, despite hashed distribution, a node becomes overloaded. However, such relocation should preserve hash locality by bumping records to nodes that are nearby in the network topology. The PIMD client always accesses records starting with a request to the PIMD server indicated by the hash location however the server task may request operations on the node a record has been bumped to. In addition, since S3M nodes will fail, PIMD must allow selected data sets to be stored with sufficient redundancy for any node's data to be recovered using a method that allows the S3M highly parallel processor fabric to scalably rebuild and redistribute data after failures.

To fully leverage the capability of the system, embedded parallel modules (EPMs) are run on the S3M active storage fabric concurrently with the PIMD server. The PIMD server and the EPM using the PIMD client are both parallel jobs executing on the S3M platform. The EPM using the PIMD client connects to the PIMD server using standard remote direct memory access (RDMA) protocols such as the OFA verbs[8]. The efficiency of connecting interoperating parallel programs, such as PIMD client to server, determines how fine-grained EPMs can usefully be in the ASF model. Once connected to the PIMD server an EPM task may access records in a PDS by key, using parallel iterators, or access all records stored on the local node. EPMs use the PIMD client and can be implemented using MPI (or an alternative parallel environment) for application specific communications and both forms of communications exploit the S3M network. S3M platforms allow EPMs to be highly scalable parallel programs. Creating environments to allow execution of function inside storage is not a new concept[11, 9], however, the S3M network allows the embedded function to be scalable and to have access to the entire data set rather than a disk block. This enables wider array of data formats and functions to be storage embedded.

ASF in practice has a minimum of two interoperating parallel jobs, the server and at least one client. The client can run on S3M hardware or a conventional server. The model anticipates an even greater level of parallel multitasking with PIMD storage as primary means of communications between interoperating parallel programs. In order to support safe, multiuser, parallel



(a) Memory to memory "Indy" sorts of various sizes on Blue Gene/L

(b) Pointer chasing through in-memory array

Figure 1: Micro-benchmarks run on Blue Gene/L showing performance relevant to data intensive applications.

multitasking jobs in ASF, we use a Linux kernel on the S3M nodes including on our BG/P prototype platform. While our simple Linux environment may limit absolute scalability, it allows our prototyping effort to exploit standard packages (i.e. OFED, MPI, GPFS), and significant progress has been made toward enabling Linux to function in this capacity [14]. EPMs are currently written as standard MPI programs which use the PIMD client to access active storage and we get significant leverage from the use of open source software.

While PIMD datasets might be accessed only by a single application embodied as an EPM, the ASF model anticipates that applications will be constructed using multiple interoperating EPMs which share PIMD datasets much as Unix programs interoperate using files. Interoperating EPMs may access PIMD datasets either directly using the PIMD client or through middleware interfaces. In the latter case, a software adapter layer uses the PIMD client and is cognizant of middleware specific record formats. Supporting interoperating EPMs allows the modularization of application solutions and encourages the reuse of modules. This modularization will allow a division of labor between those who implement EPMs, typically an HPC programming activity, and those who develop applications that use interoperating EPMs, typically a scripting or more user-oriented activity.

ASF achieves standard programming interfaces when integrated with host based middleware packages and can allow users of those packages to exploit S3M platforms transparently. By modifying a middleware package to use ASF to store objects it is generally possible to offload hot spots from the host into the active storage while non-EPM modules continue to function as before. A host based module will use the modified data access path to fetch PIMD records into the existing middleware or application data structures where they are operated on normally. Executing an EPM after such host based accesses can require the application to flush and lock data objects for consistency. For example, a package using an embedded sort EPM after host based module has accessed the target PDS must ensure that the entire PDS has been flushed out to the PIMD server before the sort EPM begins. The required interfaces are likely available since applications using disk based persistent storage need methods to deal with similar issues. Since EPMs and host based modules can interoperate using PIMD, applications can be incrementally parallelized. Non-performance critical sections of code, perhaps the vast majority of an application, can remain host based.

3. ASF ACCELERATION OF UNIX UTILITIES

The Unix file is commonly the main logical container in many storage solutions. Even with PIMD as our basic storage access method, files will be needed for operating system data and user program data (including EPMs) as well as application data. ASF has been integrated into the IBM General Parallel File System (GPFS)[13] to meet these needs and to provide the initial basic ASF accelerator platform.

GPFS has a tremendous number of useful features including distribution of data among a large number of nodes and integration with data life cycle management utilities. In order to exploit S3M platforms, we modified a research version of GPFS to access the active storage fabric using the PIMD client. The GPFS cluster accesses the entire ASF partition as a SAN or single disk image. This allows a large number of GPFS nodes to access an even larger number of ASF nodes, each

with a relatively small fraction of the total active storage. Each GPFS cluster member node uses the PIMD client to read and write file blocks. A single GPFS file system is mapped to a single PIMD PDS. The PIMD record key for each block is formed from Unix file number (inode) and block's offsets. PIMD record's value is simply the block of data. The data portion is amenable to compression.

These modifications allow GPFS to create a file system that is backed by the memory of a S3M system, which in the case of our prototype means the memory of a Blue Gene computer. On current Blue Gene machines, we run a GPFS cluster on the I/O nodes interconnected by an external Ethernet. Each GPFS cluster member uses the PIMD client to access a single, large disk image using internal connections to the compute node fabric.

Embedded Process Modules which exploit the S3M computational capabilities can be invoked in the active fabric and access GPFS files given the inode number. There are two options in providing direct access to the GPFS PDS for EPMs. First, the EPM can run as trusted code which is allowed to open the PDS directly thus bypassing all file system security. Second, a light weight GPFS specific client could provide file system access to PIMD that respected GPFS security. We have explored EPMs for Active GPFS using the first method.

We have prototyped common Unix utilities for Active GPFS including `grep` and `rm`. We have also implemented an application specific EPM which generates Indy sort records[6] directly into an Active GPFS file as well as an EPM to carry out the parallel sort of that file. The objective was to benchmark equivalent function to the Unix script `gen 1TB >start; grep AAAAAA start > t1; sort t1 > t2`. Performance results from this effort are in Table 1 which show significant speedups over the equivalent command line utilities for smaller datasets and show the ability to perform those commands on datasets of sizes that are effectively impossible to compute on a single server.

The Active GPFS test driver was hand coded, however future work could include an ASF aware shell which would automatically select the EPM utilities when input operands are large and in the active file system. Utilities that had not yet been parallelized would interoperate with EPMs but would require pulling file blocks over to the host via normal file system accesses, for standard processing.

4. ASF RDBMS ACCELERATION

A Relational Database Management System (RDBMS) such as IBM DB2 can exploit ASF and S3M machines for transparent acceleration. We are currently completing an ASF DB2 prototype which exploits DB2's Federated Wrapper interface for external data sources. Since the DB2 wrapper interface allows only read operations, we need a collection of components in order to achieve fuller RDBMS functionality using ASF. This conglomeration we affectionately call 'Frankenbase'.

The main objective of Frankenbase is to allow DB2 to offload relational operations on ASF datasets through

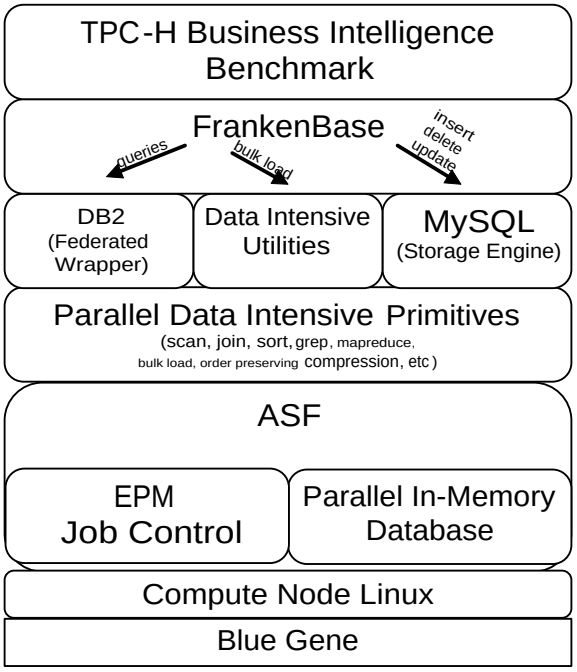


Figure 2: Diagram showing the software architecture of "Frankenbase" and how different subsets of function are handled (queries, inserts, bulk loading).

the federated wrapper interface[12]. The DB2 federated wrapper interface understands what the external data source is capable of through a query/response interface and, assuming at least basic access to table data, allows DB2 to 'compensate' for any missing function using a simple row retrieval. This allows us to incrementally add parallel capability to Frankenbase starting from row retrieval and predicate/projection scans up through multi-table joins. DB2 parses the SQL, builds an execution plan, offers to offload work, and compensates for missing offload function when required.

As Figure 2 shows, there are several major components to Frankenbase and they require a significant amount of integration code to stitch together a fully functional RDBMS. The ASF Relational Layer (ASF-RL) is the infrastructure required to enable Frankenbase to store relational tables in PIMD datasets and carry out embedded relational operations. Figure 3 provides a view of the various components of Frankenbase laid out on the S3Mplatform with a connected host. ASF-RL stores a single relational table in a single PIMD PDS and a single relational row in a single key/value record. The binary format of a row is accessed using an accessor class with methods supporting field-level access without unpacking rows to records. Table definition metadata that defines the behavior of this class is represented in a flat serialized format that can be distributed for parallel projections and joins on persistent tables or intermediate row streams. This serialized format can be generated from a user defined relational table definition via the SQL CREATE TABLE command. The tuple format is most often accessed just once and propagated into

Function (Measured at Host shell and break out of EPM components)	BG/ASF 512 nodes	pSeries p55 (1 thread)
Total host command time: create Indy File	(22GB) 22.26s	686s
Total host command time: unix grep of Indy File	25.1s	3180s
File Blocks to PIMD Records (Embedded grep):	7.0s	
Grep core function (Embedded grep):	2.5s	
PIMD Records to File Blocks (Embedded grep):	7.6s	
Total time on Blue Gene/ASF fabric (Embedded grep):	18.8s	
Total host command time: sort output of grep:	60.41s	2220s
File Blocks to PIMD Records (Embedded sort):	2.9s	
Sort core function (Embedded sort):	12.2s	
PIMD Records to File Blocks (Embedded sort):	15.8s	
Total time on Blue Gene/ASF fabric (Embedded sort):	54.7s	

(a) 22 GB synthetic "Indy Benchmark" file on 512 node BG/ASF partition vs. pSeries p55

Function (Measured at Host shell and break out of EPM components)	BG/ASF 512 nodes	BG/ASF 8192 nodes
Total host command time: create Indy File:	(22GB) 22.26s	(1TB) 197s
Total host command time: unix grep of Indy File:	25.1s	107s
File Blocks to PIMD Records (Embedded grep):	7.0s	18.1s
Grep core function (Embedded grep):	2.5s	5.1s
PIMD Records to File Blocks (Embedded grep):	7.6s	19.9s
Total time on Blue Gene/ASF fabric (Embedded grep):	18.8s	50.2s
Total host command time: sort output of grep:	60.41s	120s
File Blocks to PIMD Records (Embedded sort):	2.9s	6.8s
Sort core function (Embedded sort):	12.2s	14.8s
PIMD Records to File Blocks (Embedded sort):	15.8s	22.4s
Total time on Blue Gene/ASF fabric (Embedded sort):	54.7s	66.55s

(b) 22 GB synthetic "Indy Benchmark" file on 512 node BG/ASF partition vs. pSeries p55

Table 1: Breakdown of benchmark results for 3 runs. Each benchmark involved generation of a synthetic "Indy Benchmark" data file, a `grep` for records containing "AAAAAA" (yielding about one third of the original data), and sorting the output from `grep`.

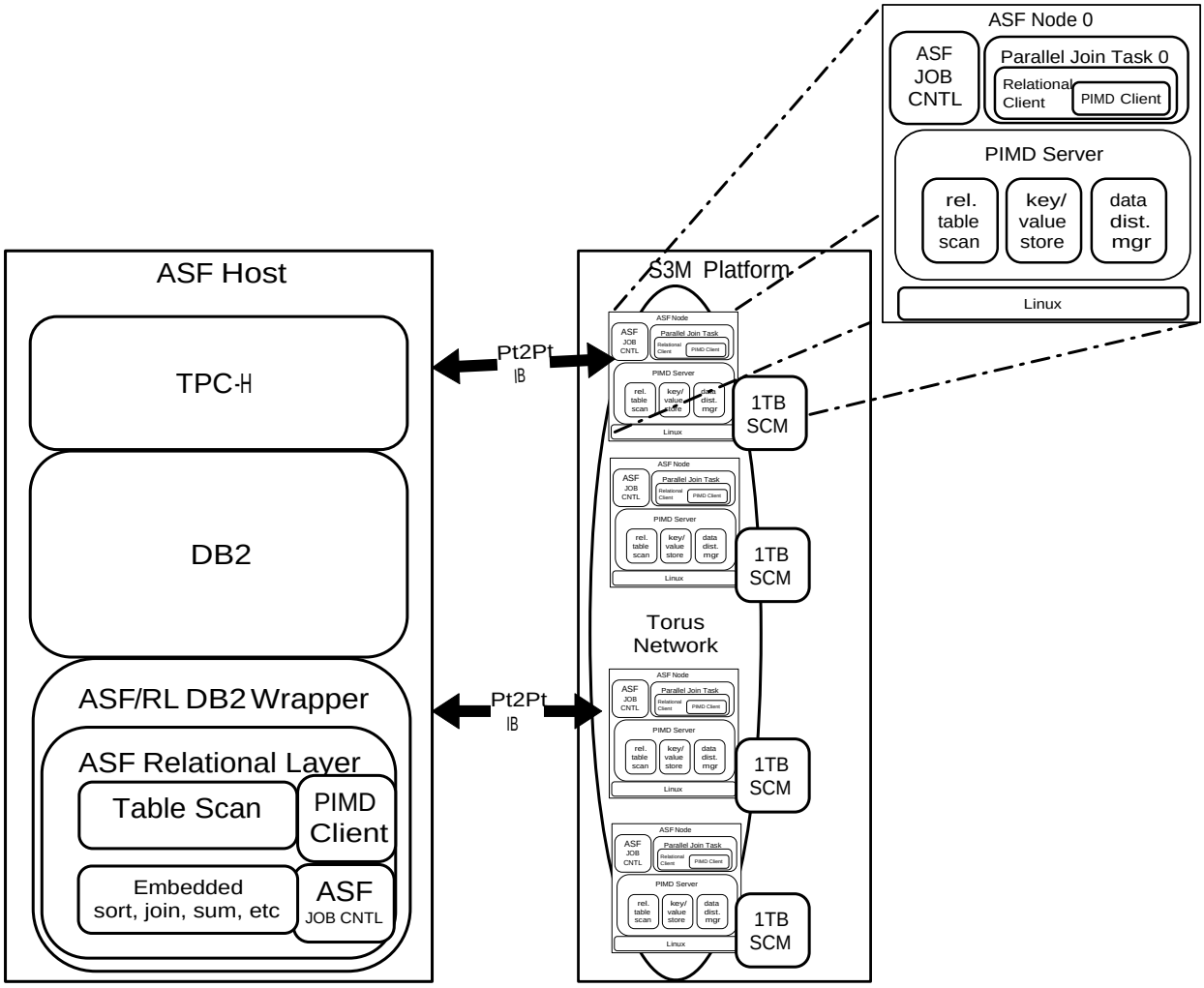


Figure 3: Schematic of ASF and ASF-Relational Layer components for RDBMS acceleration.

the required contexts for performing an operation on an ASF-RL table, for example the tasks in the parallel join EPM. The formats for tables (metadata) are managed in a single Frankenbase server process.

The primary EPMs in the ASF-RL required by Frankenbase are the table scan and relational join. The ASF-RL join EPM is a parallel program which supports a simple data flow graph environment in which we have implemented multi-table joins as a sequence of hash-joins. The first stage of the pipeline performs a two table join and propagates resulting tuples to the next stage which joins them to the third table and so on. For joins on large datasets, the join EPM may be compiled with knowledge of the specific row format definitions of the operands, reducing overhead in accessing fields in the binary representation of rows during projections and joins. When the ASF-RL join is used for DB2 acceleration, this compilation would happen as part of the query process.

Our immediate objective is to measure a significant portion of the TPC-H[10] benchmark using Frankenbase. TPC-H tables will be defined and loaded using outside utilities and the TPC-H queries will go through DB2. In principle, a user of Frankenbase will realize the full function of a RDBMS running on the host which transparently exploits the S3M platform to accelerate operations on large datasets.

5. CONCLUSIONS AND FUTURE WORK

Petascale computational resources will drive sharp increases in scientific data volumes which must be stored and manipulated. Maturity of tools and methods among communities of computational scientists will drive ad-hoc access to large data sets as multiple lines of investigation repeatedly mine these data sets. Trends in hardware technology, such as the approaching end of processor clock frequency scaling and cost effective storage class memory solutions, motivate a reexamination of the classic relationship between storage and processing. A system addressing petascale data and storage challenges should do so without requiring its users to become heroic petascale parallel programmers. The Active Storage Fabrics model addresses these challenges in a scalable way that leverages the value of current end user skills.

We have outlined the Active Storage Fabrics model and the S3M hardware architectural strawman which it targets. We have shown paths to transparent acceleration of utilities in two data management frameworks, a POSIX file system and a relational database, and how storage embedded utilities and legacy utilities can interoperate. We have created prototype systems based on the ASF model using current Blue Gene/L and Blue Gene/P systems as provisional S3M platforms, noting the lack of persistence. We presented results supporting the case for Blue Gene's capabilities as a S3M platform and the ASF model as a path to exploiting those capabilities. We have described our current work aimed at exploiting the ASF prototype for transparent relational database acceleration. Our early results demonstrate the feasibility and effectiveness of the Active Storage Fabrics model in addressing petascale data and storage challenges. Future work includes refining and extending our ASF prototype on the Blue Gene/P supercomputer in preparation for follow-on machines.

"Tape is Dead, Disk is Tape, Flash is Disk, RAM Locality is King"–Jim Gray[7]

6. REFERENCES

[1] Nsf awards $20 million to sdsc to develop "gordon", November 2009. `http://ucsdnews.ucsd.edu/newsrel/supercomputer/11-09Gordon.asp`.

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 1–14, New York, NY, USA, 2009. ACM.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst., 26(2):1–26, 2008.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, pages 205–220, New York, NY, USA, 2007. ACM.

[5] R. Freitas and W. Wilke. Storage-class memory: The next storage system technology. IBM Journal of Research and Development, 52(4/5):439–448, 2008.

[6] `http://sortbenchmark.org`.

[7] J. Gray. Tape is dead, disk is tape, flash is disk, RAM locality is king. `http://research.microsoft.com/en-us/um/people/gray/talks/flash_is_good.ppt`, 2006.

[8] `http://www.openfabrics.org`.

[9] J. Piernas, J. Nieplocha, and E. J. Felix. Evaluation of active storage strategies for the lustre parallel file system. In SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pages 1–10, New York, NY, USA, 2007. ACM.

[10] M. Poess and C. Floyd. New tpc benchmarks for decision support and web commerce. SIGMOD Rec., 29(4):64–71, 2000.

[11] E. Riedel, C. Faloutsos, G. Gibson, and D. Nagle. Active disks for large-scale data processing. Computer, 34(6):68–74, Jun 2001.

[12] M. T. Roth, M. Arya, L. Haas, M. Carey, W. Cody, R. Fagin, P. Schwarz, J. Thomas, and E. Wimmers. The garlic project. SIGMOD Rec., 25(2):557, 1996.

[13] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In FAST

'02: Proceedings of the 1st USENIX Conference on File and Storage Technologies, page 19, Berkeley, CA, USA, 2002. USENIX Association.

[14] http://www.mcs.anl.gov/research/projects/zeptoos/.