RC24586 (W0806-062) June 16, 2008
Computer Science

IBM Research Report

Toward a General Parallel Operating System Using Active Storage Fabrics on Blue Gene/P

Blake G. Fitch, Aleksandr Rayshubskiy, T. J. C. Ward*, Robert S. Germain
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

*IBM Software Group
Hursley, UK

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

1 Introduction

We propose an architecture for a General Parallel Operating System (GPOS) to make the task of efficiently exploiting Massively Parallel Processing (MPP) machines easier. Technology trends, particularly the apparent end of semiconductor frequency scaling and the limits to symmetric multiprocessor scaling, are driving a general interest in extending the reach of MPP machines. GPOS aims to enable the reuse of parallel utilities by end users in much the same way that Unix enables reuse of serial utilities via files and pipes. If GPOS is successful, it will be possible to efficiently exploit MPP machines like Blue Gene[5] using skills ranging from programming in scripting languages to MPI. In contrast to some efforts to use distributed computing for commercial applications, GPOS leverages lessons learned about how to achieve scalability from the HPC world. Large portions of the GPOS effort involve innovative integration and configuration of preexisting, successful parallel software packages and technologies which somewhat mitigates the risk of the inherently aggressive GPOS goals. Work on GPOS has produced an early prototype and while there is clearly significant work and exploration ahead, the early results are promising.

Our work has focused on a shared storage model with embedded parallel processing which is an approach we call Active Storage Fabrics (ASF). ASF uses a Parallel In Memory Database (PIMD) to allow the persistence of structured user data between concurrent and/or successive parallel job steps. PIMD is a client/server key/value database with an interface like gdbm or BerkeleyDB. PIMD servers, running on each node of the Blue Gene/ASF partition, and accessible from both embedded and external applications using a PIMD client library. An active in-memory file system, is created by modifying GPFS to use PIMD as backing store. This enables rapid access to files including embedded application executables and their operands. Active storage embedded parallel applications run in "virtual partitions", which are mapped to a subset of the physical partition hardware resources, run concurrently or successively, and are able to share data via PIMD and/or GPFS. By allowing persistent (in-memory, between jobs), structured storage shared by embedded parallel modules, reducing overheads, and meeting standard interfaces, we will enable the integration of Blue Gene with standard enterprise and scientific IT infrastructure.

The GPOS objective is an integrated system that looks and feels to the end user on a Front End Node (FEN) like a standard general purpose server rather than an HPC supercomputer or cluster. Many users will experience GPOS as an accelerator attached to a platform they already use, for example a Unix/Linux machine running commands and libraries they already know. At the same time, it provides an environment for the skilled parallel programmer to see his work reused as easily as a Unix utility. This generality will have some cost in overhead compared to writing a single, monolithic HPC MPP solution just as Unix was less efficient than MS/DOS for some programs; our challenge is to drive those overheads down using what we have learned from scaling HPC programs. The lower the overheads, the more fine grained the parallel modularization may be and the larger the number of MPP nodes that may be used in a single work stream. Eventually, modularity and efforts to reduce overhead will allow GPOS to be used routinely to create solutions more efficiently than today's HPC or cluster programming approaches.

The remainder of this paper discusses the approach we are taking to realizing GPOS and highlights some of the areas we are exploring.

2 Active Storage Fabrics – A Shared Storage Model for GPOS

In the 1974 paper, "The Unix Time-Sharing System", the authors wrote "The most important job of Unix is to provide a file system."[4] Since then, "Everything is a file" has been the most prevalent paradigm in the operating system world. The file abstraction allowed programs to interoperate and for applications to be built of processes sharing data via pipes and through the Unix file system. Although there are parallel versions of the Unix file system, a file itself can be a serializing structure. GPOS needs a mechanism for parallel programs to interoperate in a way similar to Unix processes. The shared storage model supported by the Active Storage Fabrics (ASF) approach described below provides such a mechanism.

ASF manages the MPP distributed main memory as shared storage which supports embedded parallel processes in much the same way as Unix manages the caching of file system pages in main memory. The shared storage is implemented as a Parallel In-Memory Database (PIMD) which is a key/value database similar to Berkeley DB^{TM} or gdbm. The utility of a record oriented organization for datasets operated on by parallel programs has been recognized by Google[1], but has historic roots in systems like IBM VSAM^{TM}[2].

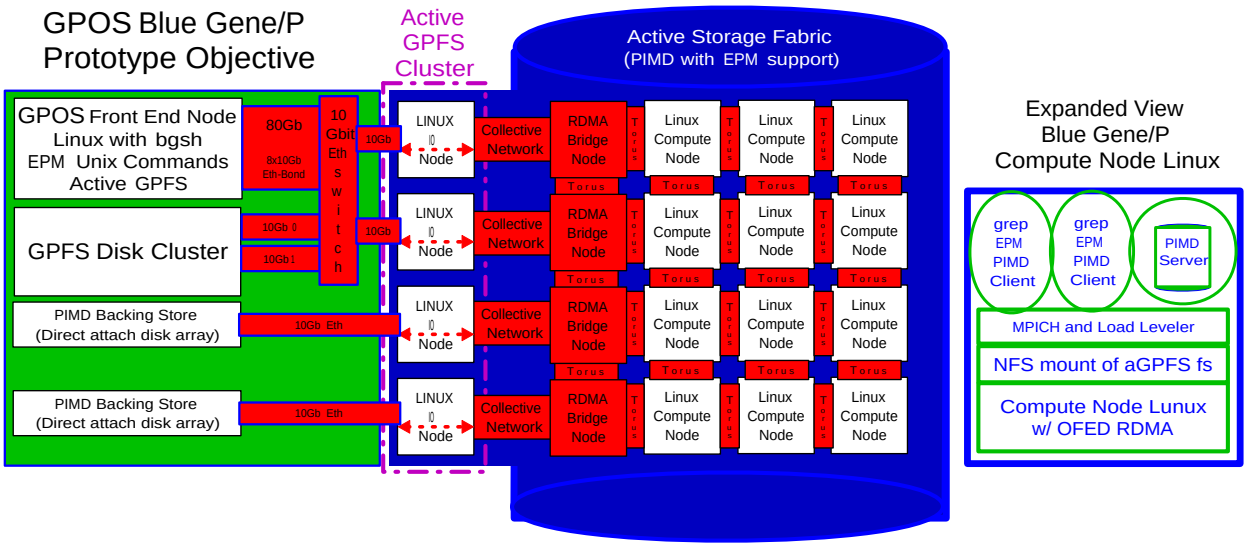

Figure 1:

The basic ASF concepts are:

- Manage MPP main memory as scalable shared storage with embedded processing capability
- Use this "Active Storage Fabric" to support parallel file systems like GPFS^{TM} and potentially other data management systems such as VSAM^{TM} or RDBMS systems (DB2^{TM}, MySQL^{TM}) using an underlying key-value parallel in-memory database
- Accelerate host system (or Front End Node) jobs with ASF:
  - Create libraries of reusable, low overhead Embedded Parallel Modules (EPMs)
  - EPMs load from and work with objects in the ASF supported file systems or data bases
  - ASF datasets are accessed via standard interfaces from EPMs and host programs
  - EPMs smoothly interoperate with each other and legacy host programs (enabling pipelining of parallel job-steps)
- Integrate Active Storage Fabrics with information life-cycle and other industrial IT management products

The Active Storage Fabrics approach could be taken with a wide variety of MPP machines. We are focused on the Blue Gene family of super computers which represent processor complexes (memory, networks, CPUs) that are hosted by standard Linux control and front end machines. For HPC processing, Blue Gene runs a light weight kernel on the compute nodes and Linux on IO nodes. ASF uses Linux on compute nodes, i/o nodes, and external Front End Nodes as shown in Figure 1. In order to exploit the Blue Gene networks we are exploring the modification of iWARP protocols to use a transport layer capable of running over BG/P hardware and IP networks. This hardware/software platform shares some features with large Infiniband clusters but achieves unprecedented density and power efficiency. If we imagine future MPP processor complexes, they may well look like racks of Blue Gene/P shrunk to fit into a standard server or even personal computer.

PIMD is a parallel, client/server key/value database that focuses on in-memory data organization but may be backed by persistent storage. A Blue Gene partition used for ASF work will have a PIMD server running on every compute node. PIMD is a record based data storage system that organizes groups of records into Partitioned Data Sets (PDSs) which are distributed across a group of PIMD servers. PIMD keys and values may be fixed or variable length, configurable on a per PDS basis. The distribution is completely

controlled by the PIMD and any record may be moved by rebalancing operations or Cuckoo hashing[3] since on a distributed memory MPP machine each node has a very small (1/P) fraction of the total memory. The PIMD client library may be used on the compute node fabric or from external machines, in serial or parallel jobs. When the PIMD client connects to the PIMD server group, it is given the information, including data distribution hash function, required to address requests to the individual server process likely to be able to process this request most efficiently. PIMD respects Unix-like access permissions for PDSs.

Embedded Parallel Modules (EPMs) are executed in a Virtual Partition (VP) which is the container for parallel job execution and is analogous to the Unix process. The VP concept separates setting up an MPI job, including task connections, from loading and running an application. A VP is a parallel job initiator, preconfigured with all connections and task processes, that is available to dynamically load and run an application. VP hardware resources may overlap but GPOS gang scheduling should avoid over subscription of hardware resources when running parallel jobs. A VP runs one parallel program at a time and is the unit of scheduling and resource allocation. A VP is comprised of one or more Linux processes on nodes of the MPP machine which are numbered from 0 to N-1 and are implicitly connected. The VP has a lead process which interacts with the GPOS scheduler to load, run, suspend, exit, and cleanup. The GPOS work scheduler aims to schedule EPMs efficiently based on availability of operand datasets and hardware resources.

EPMs may exchange data by running to completion and leaving a file or PDS in PIMD for the next EPM. If the EPMs are Unix file utilities where one EPM pipes data to another EPM, and this is recognized by the calling shell, intermediate data may be passed directly between EPMs as a PDS rather than paying to create a Unix file. In some cases a Unix pipe operation where one parallel program inserts records into a PDS for another to read without storing them in PIMD would be advantageous. PIMD will offer special services to allow the reader to tell when the writer has closed its access to the PDS and a RetrieveAndDelete operation to allow the reader to drain the dataset as it is created. Using PIMD as a communication mechanism between EPMs has overheads in communications and temporary storage but it allows EPMs to be independent in terms of data distribution, contributing to generality and reusability.

ASF requires sharing data between FENs, compute nodes, and standard information life cycle management subsystems. In many cases these systems use a Unix file system to share data including program inputs, outputs, and executables. Rather than figure out how to make an operating system work natively with PIMD instead of a Unix style file system we modified IBM's GPFS^{TM} to use PIMD instead of a disk or SAN. On BG/P, a GPFS^{TM} cluster is then run on the BG/P IO nodes so that NFS can be used to mount the PIMD backed GPFS^{TM} file system on the compute nodes to, for example, load programs. The modified GPFS^{TM} cluster on Blue Gene sees a single PIMD PDS as a large single disk image which GPFS^{TM} conveniently manages concurrent access to. The key used to store GPFS^{TM} blocks is not the offset into this disk image but rather the inode and file offset of the stored block. This allows file blocks to be located rapidly once the inode is known by directly accessing the PIMD table backing the GPFS^{TM} file system. During such operations, GPFS^{TM} must lock the file to avoid breaking proper Unix file semantics. Such fast access methods allow large node count EPMs to open and access file data efficiently. Currently this client is a light weight user library but to ensure credential checking, this interface will be eventually handled as a system call which uses the PIMD client.

There are two approaches to the persistent storage required by GPOS. First, PIMD may be directly backed by a larger persistent store with the addition of operators to handle journal modifications, demand retrieve rows, and swap tables between memory and disk. Alternatively, GPFS^{TM} can be used for file system mirroring or hierarchical storage interactions. The former would allow persistence of PIMD tables directly where the latter requires that data be placed into Unix file format before being moved to persistent store. These techniques will allow GPOS exploit the ASF model for far more storage than the main memory of the MPP machine.

Like the file system did for Unix, ASF provides a framework for tying together the components of GPOS. GPOS starts with the Blue Gene/P MPP machine with its common clock, high performance networks, and power efficient stateless nodes as a model of future dense processor complexes. GPOS uses Linux as the compute node kernel which provides GPOS the primitives to support multiuser, multitasking virtual memory parallel jobs rather than a single monolithic MPI job which is the MPP and Blue Gene norm. GPOS will use standard RDMA interfaces to support both the Parallel In-Memory Database and MPI for EPMs. GPOS uses the IBM General Parallel File System (GPFS), backed by PIMD to integrate ASF with both the Linux compute node fabric and external Linux nodes supporting end users and persistent storage. GPOS FENs

will run a modified Unix shell which recognizes when operands are on the ASF file system and invokes EPMs rather than standard, host based Unix utilities. Initially, the EPM Virtual Partition execution environment will be created using MPI jobs dispatched onto the compute node Linux cluster using IBM Load Leveler.

3 Using GPOS

The GPOS user works on a Front End Node which can be a server or personal computer which has been configured to attach to an ASF MPP partition. The interactive user, script, or controlling program uses the MPP machine transparently by using commands and libraries which move datasets onto the Active Storage Fabric and trigger embedded parallel modules. Roughly speaking, this is the "accelerator model". The GPOS user environment is configured to have access to the ASF backed GPFS^{TM} file system and to have access to a job queuing system. Active file system based EPMs that have been implemented are `sort`, `grep`, `rm` and `generate_indy_sort_data`. Other host based programs such as database engines may access ASF storage directly using the PIMD client and trigger PDS based EPMs such as "parallel join".

There are two major classes of programming in the GPOS environment. First, end users can write EPMs which is about as difficult as writing any MPI parallel program. Second, end users may invoke EPMs as utilities or libraries from a GPOS FEN using a program that controls work flow by invoking EPMs. As EPM overheads decrease, parallel modules may become finer grained. In certain cases, streaming interactions between parallel modules are possible.

A General Parallel Shell (gpsh) along with standard libraries which are accelerated by the ASF will make GPOS user programming similar to Unix. Command line operations may be accelerated by EPMs along with scripts and standard library calls. The gpsh shell will automatically detect cases where operands are on the ASF file system and invoke the appropriate EPM's wrappers as needed. The shell will also detect when intermediate data isn't stored to a named file, for example when grep is piped into sort, avoid the overheads of serializing a PDS to a file by telling the EPMs to directly communicate through a PDS. Since Unix utility EPMs can convert to/from Unix files for named datasets, EPMs and FEN based commands can interoperate smoothly.

GPOS enables parallel programs to interoperate in a fashion analogous to interoperating Unix processes. Although significant performance advantage will be available to those who want to write an EPM, we expect many users will compose applications by using interoperating parallel versions of Unix file utilities. For example, the Unix command line `grep XYZ input | sort > output` is interpreted as grep reading a file called input, finding the lines containing the string "XYZ", passing them via a pipe to the sort program which reads and sorts lines on stdin until it is closed and then writes a file called output. The intermediate dataset passed between grep and sort never needs to be realized as a Unix file. In GPOS, an output dataset is created in the ASF. Currently grep and sort are coded as EPMs that expect to read input from and write output to the PIMD backed GPFS^{TM} file system in the ASF. The data that passes from grep to sort is realized as a Unix file at some cost. The limited GPOS prototype on BG/L was able to achieve significant speed up over FEN based grep and sort EPMs using a 512 node partition and to scale to operate on terabyte sized files using an 8192 node partition.

4 Conclusions and Further Research

We have described an approach to a General Parallel Operating System based on Active Storage Fabrics. We have implemented an early prototype of some ASF functionality on Blue Gene/L with promising results including creating a 4TB (Terabyte) active GPFS^{TM} file system and executing Unix grep and sort as parallel embedded programs. We are currently moving from BG/L to BG/P and making a number of changes including using Linux as the compute node kernel and improving PIMD. This platform will allow further development of parallel versions of standard Unix utilities for active GPFS^{TM}, a shell able to transparently invoke these utilities, and explorations into an active relational database storage engine.

References

[1] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[2] Mary Lovelace, Dave Lovelace, Rama Ayyar, Alvaro Sala, and Valeria Sokal. *VSAM Demystified*. IBM International Technical Support Organization, September 2003. SG24-6105-01.

[3] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *ESA '01: Proceedings of the 9th Annual European Symposium on Algorithms*, pages 121–133, London, UK, 2001. Springer-Verlag.

[4] Dennis M. Ritchie and Ken Thompson. The unix time-sharing system. *Commun. ACM*, 17(7):365–375, 1974.

[5] IBM Blue Gene Team. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development*, 53(1/2):199–219, January/March 2008.