Shared Memory On Disk: SHMOD
Overview

November, 2002

Sarah Anderson
David Porter
P. R. Woodward

Laboratory for Computational Science & Engineering
University of Minnesota

This is a simple scheme for accomplishing the communication and coordination necessary to execute tasks comprising a numerical simulation program on a cluster of computers. It is called "SHMOD" because it simply implements a minimalist globally shared memory, with disk memory and file systems as its basis. It is not necessarily built on a true shared file system, but on local system disks augmented with a high performance communication network and a user-level library. We find it useful to consider that there are two types of shared memory required for applications: one requiring high bandwidth non-blocking bulk transfers, essentially remote file access, and the other providing atomic access to a small amount of global control data for task coordination. Control data can be handled either with an explicit "task manager" process, or, as we have done, by cooperating processes reading, or writing (and locking as necessary) a global data base provided by a standard data-base server.

The computational fluid dynamics simulations we have performed need a domain decomposition for parallelization on a cluster, that is, subdivision of the global problem into pieces. These pieces, data consisting of physical state variables such as fluid density, pressure and velocities, are advanced in time using appropriate numerical solutions. When the domain decomposition is extended to bundling a node's domain data with all other required data (such as 'halo', or fake-zone data from other zones) we have a completely self-contained unit of work which can be assigned to <u>any</u> node. High bandwidth bulk data movement permits each participating node application to pre-fetch, update and write-back a pipeline of complete context data updates. If the context data can also be seen by any node, we therefore have a framework which admits not only parallelism, but load-balancing, fault-tolerance and "out of core" computing.

Both sorts of remote data access is exposed to the application programmer as the ability to read and write unstructured "blobs of bytes". To the application, these are used as structures (C) or common blocks (Fortran). There is a shared name-space for each type of shared object, the bulk-i/o objects and the control objects.

The first call that must be made by a node participating in SHMOD is one which sets up communication with the data base server and run-specific data base. Ideally, it would be matched with a call to stop_thing(), but this is optional, as both SHMOD and mySQL are capable of "cleaning up after themselves".

int start_thing(char *host, char *user, char *pw, char *dbname)

Returns 0 if all OK.
-1 Failed to connect to db server
-2 Could not start io subthread

Example: (In fortran the call is the same, just omit the ";")

```
ierr= start_thing( "user02.ncsa.luiuc.edu", "sanderso", "", "sea01" );
```

char *dberror()
If start_thing returns -1, the mySQL error message describing why can be helpful for diagnosing problems with start_thing.

void stop_thing();

Terminates the connection to the data base server and shuts down subsidiary t threads() for performing network or disk I/O.

To get and put named blobs of bytes which are used for small amounts of control data, these two blocking calls are provided:

int getglob_thing(char *name, void *addr, unsigned nbytes, unsigned *timestamp)

int putglob_thing(char *name, void *addr, unsigned nbytes)

The name of a "glob" is a symbolic name useful to the programmer. A data-base table called "globs" records the name, and size of the blob.

It is often useful to have a consistent (i.e. global) idea of the time, for node decisions on task assignment timeouts and the like. The data-base servers idea of the current time, accurate to at least to the second, is returned in the timestamp.

For work sharing constructs, a typical use would require atomic access by an individual node to a glob. For example, (omitting the good practice of checking error return codes!) we would write:

```
/* lock glob table */
ierr= getglob_thing( "workinfo", &workinfo, sizeof(workinfo), &thetime)
/* figure out what to change in workinfo based on its contents... */
ierr= putglob_thing("workinfo", &workinfo, sizeof(workinfo) )
/* unlock glob table */
```

The provided calls to lock the data base server for globs are:

int lock_thing(char *name)
int unlock_thing(char *name)

where the name is an arbitrary lock name decided on by the programmer.
The set of calls to accomplish high-bandwidth non-blocking I/O is perhaps best thought of as what they are, really: remote file access.

Error returns:
-2 The named object does not exist in the data base
-1 Cannot obtain an I/O process request (too many)

0 Success.
>0 in the case of wait_thing only, the ultimate number of bytes actually read or written.

In order to read or write an object, it must be first created in the data base.This need be done only once (not in each program,and not on each access), though it can be “re-created” if desired to override the hostname or root directory.

int create_thing(char *name, char *hostname, char *directoryname, unsigned nbytes)

int delete_thing(char *name)

int read_thing(char *name, Offset_t offset, void *addr, unsigned nbytes, int *iostatus)

int write_thing(char *name, Offset_t offset, void *addr, unsigned nbytes, int *iostatus)

int wait_thing(int iostatus, int howlong)

At any time after a thing is created, the current information can be retrieved from the data base server with this call. Note that only “name” is an input parameter to describe_thing, the rest are all returned from a query.

int describe_thing(char *name, char *hostname, char *directoryname, unsigned *nbytes, unsigned *checksum, unsigned *timestamp)

This call retrieves all matching things (up to a maximum of *nresults)* based on the first element of the passed items, which are now understood to be arrays. This would be useful for retrieving all things on a host, or all objects satisfying a wildcarded name. Note only the description of the thing is returned; if an I/O is intended, one of the things should be selected and passed to read/write_thing.

int query_thing(char *name, char *hostname, char *directoryname, unsigned *nbytes, unsigned *checksum, unsigned *timestamp, int nresults)

For example:

```
      character(*32) qnames(10), qhosts(10), qdirs(10)
      integer qnbytes(10), qchecksums(10), qtimestamps(10)

      ...

      qnames(1)= "tilex01y01*"
c  The following match 'any' thing with the corresponding datum
      qhosts(1)= ""
      qdirs(1)= ""
      qnbytes(1)= -1
      qchecksums(1) = -1
      qtimestamps(1)= -1
      ierr= query_thing( qnames,qhosts,qdirs,qnbytes,qchecksums,qtimestamps,10)
```

```
      if ( ierr .lt. 0 ) then
c             // error processing- query failure
      else
      do i= 1, ierr
      print *, "found ", qnames(i),qhosts(i),qdirs(i),qnbytes(i),qchecksums(i),
     &                            qtimestamps(i)
      enddo
      endif
```