EGIDA DOCUMENTATION

1 Setting up Egida

All directories are relavant to my directory structure. Make the appropriate name changes if your naming convention differs.

Set up directories and soft links

`/home/user_name` is the mounted directory and is available on all the nodes. `/export/suburbia` is on the local machine. Please change them to match to any corresponding directory on your machine. You should begin in `/home/user_name` on one of the machines you will be using to run Egida. The machines used here are magha, ardra, mriga, and phalgun.

1. mkdir EGIDA
2. cd EGIDA
3. mkdir checkpoints
4. mkdir exp_distribution
5. mkdir results
6. mkdir test_programs
7. mkdir scripts
8. cd checkpoints
9. ln -s /export/suburbia/user_name magha.Dir
10. ln -s /export/suburbia/user_name mriga.Dir
11. ln -s /export/suburbia/user_name ardra.Dir
12. ln -s /export/suburbia/user_name phalgun.Dir

On **each** machine that will be running Egida, you will need to log in and do the following:

1. cd /export/suburbia
2. mkdir user_name
3. cd user_name
4. ln -s /home/user_name/EGIDA/checkpoints CheckpointFiles
5. ln -s /home/user_name/EGIDA/results Results
6. ln -s /home/user_name/EGIDA/test_programs/NPB2.3/bin/sp.A.4 sp.A.4

The last link is for the application that will be built later in this process.

We now need to make sure things are going to be found at runtime, so return to `/home/user_name` and open `.login` in emacs.

1. add /home/user_name/EGIDA/mpich/util to PATH variable
2. add /home/user_name/EGIDA/scripts to PATH variable

This will allow user defined scripts and `mpirun` to be found from any directory.

Install tar files that don't need compiling

You should be in `/home/user_name/EGIDA` to begin this section.

1. cd exp_distribution
2. copy expDist.tar.gz
3. gunzip expDist.tar.gz
4. tar xvf expDist.tar
5. cd ../scripts
6. copy scripts.tar.gz
7. gunzip scripts.tar.gz
8. tar xvf scripts.tar
9. emacs client-kill.pl and modify paths to be consistent with your own path naming convention
10. emacs remove-hung-procs and modify paths to be consistent with your own path naming convention
11. create `clean-up-files.pl` to remove all Egida files that are created (useful for crashes and clean-up after running.)

Install MPI and Egida

You should begin this section in `/home/user_name/EGIDA`

1. copy mpich.tar.gz
2. gunzip mpich.tar.gz
3. tar xvf mpich.tar
4. cd mpich
5. `./configure -arch=solaris86 -device=ch_p4 -cc=cc -c++=CC \`
   `-fc=f77 -mpe -opt=-O -lib=''-lnsl -lsocket''`

Note...the `./configure` command must all be on one line or you must use the unix script *continue* "\" to put it on two or more. We must now tell mpi which machines we want to run on. There is some confusion on what the machine file is named so for safety I've included both. We are currently in the `/home/user_name/EGIDA/mpich` directory.

1. cd util/machines
2. emacs machines.solaris86 and add one per line, left justified:
   magha
   mriga
   ardra
   phalgun

   (It is important that there is no space before the machine name and that only one machine name is on a line.)
3. cp machines.solaris86 machines.i86pc (this may not be necessary)

We can now build mpi. You should return to `/home/user_name/EGIDA/mpich`

1. make

Now we need to set up the modified ch_p4 library that will allow mpi to talk to Egida. You are still in the `/home/user_name/EGIDA/mpich` directory.

1. cd mpid/ch_p4
2. chmod -R u+w *
3. rm -f adi2init.c
4. rm -f p4priv.c
5. copy ch_p4.tar.gz
6. gunzip ch_p4.tar.gz
7. tar xvf ch_p4.tar
8. copy p4_rm.c to p4-1.4/lib/
9. copy p4_bm.c to p4-1.4/lib/
10. in emacs bring up `Makefile` and modify MPIR_HOME to be the following:
    MPIR_HOME = /home/user_name/EGIDA/mpich
    plus change -ICoyote/include to be -IEgida/include

The reason we had to change MPIR_HOME is that this sub-library was originally compiled in a different user account, therefore our configure command done earlier did not generate this makefile.

Before we compile this "new" ch_p4 library, we need to install and make Egida.

1. copy Egida.tar.gz
2. gunzip Egida.tar.gz
3. tar xvf Egida.tar
4. cd Egida/source
5. in emacs bring up `Makefile.common` and modify MPI_BASE_DIR to be the following:
   MPI_BASE_DIR = /home/user_name/EGIDA/mpich
6. in emacs bring up `Init.cc` and modify EGIDA_DIST_DIR, and EGIDA_APPL_BIN_DIR to look like the following:
   strcpy(EGIDA_DIST_DIR, "/home/user_name/EGIDA/exp_distribution");
   strcpy(EGIDA_APPL_BIN_DIR, "/export/suburbia/user_name");
7. gmake -r depend
8. gmake -r

Now that Egida has been built we can compile our special ch_p4 platform. You should return to `/home/user_name/EGIDA/mpich/mpid/ch_p4`

1. make default_all

Install the test suite

You should begin this section in `/home/user_name/EGIDA/test_programs`

1. copy NPB2.3.tar.gz
2. gunzip NPB2.3.tar.gz
3. tar xvf NPB2.3.tar

We will make the example *sp* program. You could also make *bt* or *lu* at this point although for the latter two programs you must take the "-0" option out of their Makefile.

1. cd NPB2.3/config
2. copy make.def
3. in emacs bring up `make.def` and modify MPI_BASE_DIR and FT_LIB to be the following:
   MPI_BASE_DIR = /home/user_name/EGIDA/mpich
   FT_LIB = $(MPI_BASE_DIR)/mpid/ch_p4/Egida/source/libEgida.a
4. cd ../SP
5. copy make_SP.def make.def
6. copy Makefile
7. You may need to make some modifications in the make.def to reflect the correct paths for the libraries. This is necessary for the Security library. The SSl library (also available with this package) needs to be put in the directory /home/user_name/SSLeay/x86/lib
8. in emacs bring up `Makefile` and modify any string "phoebe" to your "user_name"
9. in emacs bring up `make.def` and modify any string "phoebe" to your "user_name"
10. in emacs bring up `sp.f` and make sure the line
    call Egida_AppLoopStart ()
    is in the file about line 150.
11. make NPROCS=4 CLASS=A

Running Egida

At this point everything is set up and ready to go. We need to make sure our modified PATH variable can now find all the executables we have set up. You can either log out and log back in or do the following in `/home/user_name`

1. source .login
2. rehash

For the next part you must be on the machine that is first in your machine.i86 file. For me that would be magha.

1. cd /export/suburbia/user_name
2. mpirun -np 4 -v sp.A.4
   (you can also set variables like -ckptFreq=300 and -numFailures=0)

The program should now run. If you want to kill it in the middle of execution you would do the following:

1. hit the `^C` (control-C) until you get a unix prompt again. You may have to do this a couple of times.
2. client-kill.pl
3. clean-up-files.pl

2 Configuration of protocols

With Egida, any protocol can be instantiated from its building blocks by just assembling the blocks correctly. This is a build time (compile time) activity and has to be specified in the file ProtocolSpecificInit.cc. The major building blocks that need to be assembled are Delivered log (at the receiver, can be Volatile, Stable OR Null), Sent log (at the sender), Determinant Log (at the receiver), piggyback (vector clock or null, etc), checkpoint algorithm, orphan detection algorithm. Checkpointing frequency and flushing frequency can be specified as command line arguments.
As an example, here is the configuration of the blocks for the Receiver Based Pessimistic (RBP) protocol

```
sentLog = new EventLog(id, numProcs,
    EventStore::EGIDA_VOLATILE_STORE,
    EventStore::EGIDA_NULL_STORE,
    FlushPolicy::EGIDA_INDEP_FLUSH,
    EGIDA_SENT_LOG_MSG,
    -1, //in RBP each receive will clean the sentLog
    FlushScheduler::EGIDA_PERIODIC_FLUSH);

detLog = new DetLog(id, numProcs,
    DetStore::EGIDA_STABLE_STORE,
    DetStore::EGIDA_STABLE_STORE,
    FlushPolicy::EGIDA_INDEP_FLUSH,
    EGIDA_DET_LOG_MSG,
    -1,
    FlushScheduler::EGIDA_PERIODIC_FLUSH);

deliveredLog = new EventLog(id, numProcs,
    EventStore::EGIDA_STABLE_STORE,
    EventStore::EGIDA_STABLE_STORE,
    FlushPolicy::EGIDA_INDEP_FLUSH,
    EGIDA_DELIVERED_LOG_MSG,
    -1,
    FlushScheduler::EGIDA_PERIODIC_FLUSH);

howToCkpt = new AsyncCkpt(id,
    CkptImpl::EGIDA_FULL_CKPT,
    programName);

// No orphans for pessimistic/causal
orphanDetection = new NullOrphanDetection();

pbLog = new PbLog(id, numProcs, numFailures);
recovery = new Recovery(id, numFailures,
    detLog,
    sentLog,
    deliveredLog,
    pbLog,
    orphanDetection,
    CollectRecInfo::EGIDA_DISTRIBUTED_COLLECT_REC_INFO);

rollbackActions = new FlushOnRollback(detLog,
    deliveredLog);

rollforwardActions = new FlushOnRollback(detLog,
    deliveredLog);

ckptProtocol = new CkptProtocol(id, numProcs,
    ckptFreq,
    FlushScheduler::EGIDA_PERIODIC_FLUSH);
```

// for a protocol that piggybacks nothing
pbCkpt = new PbCkpt(id, numProcs, ckptProtocol);

3 Release Notes/README

Platform

Egida currently runs on Solaris 2.8 and it will be available very shortly on Linux as well. It currently features a non-threaded library, but very soon, a threaded library can be expected.

Current Status

Egida is a platform where any protocol can be built from building blocks. Currently, protocols have to be built with care to see that the combination of blocks yield a correct protocol.
Even though any protocol can ideally be built from Egida, we have tested this against the logging protocols which are currently in use, Receiver based pessimistic (RBP), Sender based pessimistic (SBP), Causal, Optimistic. The current status of Egida is that it works well and is stable with RBP, SBP and Causal and it is reasonably stable with Optimistic.

Known Bugs

- After a number of failures and recoveries, Egida, with Causal logging gives the error "determinant not found in stable or unstable". This is probably due to some race condition and needs to be taken care of.
- With the Optimistic protocol, with multiple failures of processes on different nodes, gethostbyname() fails to return the right answer on subsequent failures after the first one. Hence if there are multiple failures on different nodes, then, the subsequent failures don't recover.

Security with Egida

Egida currently has the code for security as well, but it has not been tested. It can be turned on OR off by setting one of the global variables in one of the header files (CVR - which one). The effect of having the security code in Egida is that it now has to be built with the SSL library.

Statistics/Loops Counters

Egida code has been instrumented to record the statistics, the time necessary for each task. It also prints out the progress frequently by printing out the loop numbers.

Portability

Since some changes have been made to the p4 library, Egida cannot be moved without changes to any other mpich version. That way, it is not entirely portable across versions of mpich.

4 Future Work/Ongoing tasks

- Currently, each node has a failure detector running on it. This failure detector detects the failure of any process running currently on that node. This however does not consider the scenario of power failures where the complete node goes down instead of just the process. In order to cover such failures, we plan to rearchitect the failure detector to be centralized so that it detects the failures of the processes on all the nodes.
- The Egida library is non threaded as of now and will be threaded in the next version.
- The current Egida libabry runs on Solaris 2.8. We hope to make available the next version of Egida on Linux.

- The next version of Egida will be security enabled and can be turned on or off during build time.
- Currently a lot of care has to be exerised to build the protocols from the building blocks of Egida. We plan to have a protocol builder with a user friendly front end to assist validate the correctness of the protocol.
- Installation scripts for Egida will be distributed later