Exercises for Grid Application Toolkit

Part I: Getting started

The Grid Application Toolkit is installed in /opt/gathome/GATEngine. The adaptors are located at /opt/gathome/GATAdaptors. In order to set the environment variables do the following:

```
source /opt/gathome/GATEngine/gatenv.sh
```

This will export two environment variables namely the GAT_LOCATION that points to the folder where the Grid Application Toolkit engine is installed. The second GAT_ADAPTOR_PATH tells the engine the location of the adaptors to be loaded when the engine loads up.

```
echo $GAT_LOCATION
/opt/gathome/GATEngine
echo $GAT_ADAPTOR_PATH
/opt/gathome/GAEngine/lib/GAT/adaptor-list
```

The GAT_ADAPTOR_PATH points to a text file that contains the names of adaptor libraries.

```
cat $GAT_ADAPTOR_PATH
```

In this exercise we will write GAT programs to move files between resources using the GridFTP adaptor that uses the gsiftp protocol to transfer files and to submit jobs using the GAT. In order to test file transfer we will as before create files:

```
dd if=/dev/zero of=./testfileYOURNAME bs=1M count=10
```

Again to check the size

```
ls -lh ./testfileYOURNAME
```

PART II: GAT Program to transfer files

Make a directory that will house your gat programs:

```
mkdir YOURDIRECTORY
```

```
cd YOURDIRECTORY
```

Now lets type the C Program that will copy file between machines!!!!

```
cat > copyfile.c
#include <stdio.h>

#include "GAT.h"
#include "GATTestUtils.h"

int main (int argc, char *argv[])
{
  GATContext  context = NULL;
  GATLocation name1   = NULL;
  GATLocation name2   = NULL;
  GATFile     file1   = NULL;
  GATResult   retval  = GAT_FAIL;

  /* check for correct invocation */
  if ( argc < 2 )
  {
     printf ("\n\tUsage: %s <src> <target>\n"
             "\n\tprogram does:\n"
             "\n\t\tcp <name1> <name2>\n\n" , argv[0]);
     exit   (1);
  }

  /* initialize GAT: create context */
  context  = GATContext_Create   ();

  /* create URLs for all file names */
  name1    = GATLocation_Create  (argv[1]);
  name2    = GATLocation_Create  (argv[2]);

  /* create initial file object */
  file1    = GATFile_Create (context, name1, 0);

  /* cp <name1> <name2> */
  retval = GATFile_Copy   (file1, name2, GATFileMode_Overwrite);

  /* clean up */
  GATFile_Destroy     (&file1);
  GATLocation_Destroy (&name1);
  GATLocation_Destroy (&name2);
  GATContext_Destroy  (&context);

  return (0);
}
Ctrl-D
```

Test if your file is there…

```
cat copyfile.c
```

Next we need to compile our GAT Program. In order to do this we will copy over a Makefile that will ease the job of compiling out GAT programs. We will also need to do a grid-proxy-init to be able to use gisftp to transfer files.

```
cp $GAT_LOCATION/bin/Makefile .
make
grid-proxy-init
```

Now you will see an executable file with the name of the C source file (without the extension) that you created

```
./copyfile /home/trainingXX/testfileYOURNAME
gsiftp://172.16.82.205/home/trainingXX/testfileYOURNAMEgk2
```

.

Check on gk2 if your file was transferred to there.

PART III: Submitting Jobs

Now we will write a GAT Program that reads a command line argument that is the name of an executable and some arguments and executes it on the localsystem.

Type out/paste the following program: (The program can also be found at

```
cat > submitmyjob.c
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/utsname.h>
#include <unistd.h>
#include "GAT.h"

static GATSoftwareDescription
       create_software_description (GATContext   context,
                                    const char*  exe,
                                    int          nargs,
                                    const char** args);
static GATResourceDescription
       create_hardware_resource_description (const char* host);

int main (int argc, char *argv[])
{
  char*  exe   = NULL;
  char*  host  = NULL;
  int    nargs = argc - 2;
  char** args  = &argv[2];

  GATResult         retval   = GAT_FAIL;
  GATContext        context  = NULL;
  GATResourceBroker broker   = NULL;
  GATJob            job      = NULL;
  GATSoftwareDescription sd  = NULL;
  GATResourceDescription hrd = NULL;

  /* Call this before any other GAT call! Otherwise it will fail. */
  context = GATContext_Create();

  if ( argc < 3 )
  {
     printf ("\n\tUsage: %s <host> <binary> [args]\n"
             "\n\trun the given program on the given host.\n\n",
argv[0]);
    exit (1);
  }

  /* store away args */
  host = GATUtil_strdup (argv[1]);
  exe  = GATUtil_strdup (argv[2]);

  /* submit the job to the resource broker */
  broker = GATResourceBroker_Create (context, 0, 0);

  /* create the software description of the job to start */
  sd = create_software_description (context, exe, nargs, (char const
**)args);

  /* create a hardware resource description describing the required
job
     environment */
  hrd = create_hardware_resource_description (host);
  /* test job submission through a resource description */
  {
    GATJobID_const    jobid = NULL;
    GATJobDescription jd    = NULL;

    /* make a job description out of the software and hardware
resource
       description */
    jd = GATJobDescription_Create_Description (context, sd, hrd);

    /* submit a new job */
    GATResourceBroker_SubmitJob (broker, jd, &job);

    /* time for decent output */
    sleep (1);

    /* retrieve the GAT job id of the newly created job */
    GATJob_GetJobID (job, &jobid);
    printf ("started %s: %s\n", exe, GATString_GetBuffer (jobid));
    /* clean up memory */
    GATJobDescription_Destroy (&jd);
  }

  /* free up all allocated memory */
  GATJob_Destroy                 (&job);
  GATResourceBroker_Destroy      (&broker);
  GATResourceDescription_Destroy (&hrd);
  GATSoftwareDescription_Destroy (&sd);
  GATContext_Destroy             (&context);

  free   (exe);

  return (0);
}
/* create a software description describing the job to start */
static GATSoftwareDescription
       create_software_description (GATContext   context,
                                    const char*  exe,
                                    int          nargs,
                                    const char** args)
{
  GATSoftwareDescription sw_desc    = NULL;
  GATLocation            location   = NULL;
  GATList_String         arguments  = NULL;
  GATTable               attributes = NULL;
  GATFile                file       = NULL;
  int i = 0;

  /* create and fill the attribute table */
  attributes = GATTable_Create ();

  /* create "location" attribute" */
  location = GATLocation_Create (exe);

  GATTable_Add_GATObject(attributes, "location",
    GATLocation_ToGATObject_const (location));

  /* create and fill the "arguments" attribute */
  arguments = GATList_String_Create ();

  for (i = 1; i < nargs; ++i)
  {
    GATList_String_Insert (arguments,
                           GATList_String_End (arguments),
                           args[i]);
  }

  GATTable_Add_GATObject (attributes, "arguments",
     GATList_String_ToGATObject_const (arguments));

  /* create the software description object */
  sw_desc = GATSoftwareDescription_Create (attributes);

  /* free the allocated memory */
  GATList_String_Destroy (&arguments);
  GATLocation_Destroy    (&location);
  GATTable_Destroy       (&attributes);

  return (sw_desc);
}

/* create a hardware resource description describing the required job
   environment */
static GATResourceDescription
       create_hardware_resource_description (const char* host)
{
  GATHardwareResourceDescription hw_desc      = NULL;
  GATTable                       requirements = NULL;

  /* create and fill the requirements table */
requirements = GATTable_Create ();

  /* add required OS parameters to the requirements table */
  GATTable_Add_float  (requirements, "memory.size",  0.256f);
  GATTable_Add_float  (requirements, "disk.size",    10.f);
  GATTable_Add_float  (requirements, "cpu.speed",    1.f);
  GATTable_Add_String (requirements, "cpu.type",     "unknown");
  GATTable_Add_String (requirements, "machine.type", "i686");
  GATTable_Add_String (requirements, "machine.node", host);

  /* create the software Resource description */
  hw_desc = GATHardwareResourceDescription_Create (requirements);

  GATTable_Destroy (&requirements);

  return (GATHardwareResourceDescription_ToGATResourceDescription
(hw_desc));
}
Ctrl-D
```

next as before we will compile and link the C program using the make command. To do this simply type the make command that uses the Makefile that we copied earlier.

```
make
```

This will generate the executable file submitmyjob.

```
./submitmyjob gk1.phys.utb.edu /bin/date
```

This will submit /bin/date to the local system.

EXTRA CREDIT: Look in the folder /opt/gathome/gatinstall/examples to find many examples and “one_liners” that do basic Grid operations for job submission, logical file management and data management. Based on these write a GAT program to copy a executable file from gk2 to gk1 and then execute it.