Evaluation of United Devices (UD) Grid MP SDK

Shamsundar Ashok, Subramaniam Venkiteswaran
University of Texas at Austin

With the litany of distributed systems and architectures claiming support of grid services in today's environment, we decided to evaluate one such offering from a developer's perspective. In doing so we wished to gain and convey a novice user's view on usability and usefulness when coming from a world of standard serial and parallel architectures.

We chose United Device's Grid MP platform, and as an experiment we wished to port an existing MPI-based application that relies on effective communication of intermediate results onto this platform using UD's SDK. From this we wished to gain some knowledge and experience of an actual implementation of an architecture offering currently accepted definitions of grid services, as well as issues confronted when developing on such a platform.

The report is organized as follows. Section 1 introduces UD's Grid MP platform and SDK. Section 2 deals with the experiment candidate – the Genetic Algorithm for Maximum Likelihood (GAML). Section 3 discusses our experiment approach and results/observations. Section 4 provides conclusions based on the current state of this evaluation and future directions.

1 Brief Introduction to UD's SDK:

UD's motivation arose from the fact that there are a large number of underutilized desktop computers, whose resources if pooled together would offer comparable processing throughput as some of today's supercomputers. UD addresses this by attempting to provide a distributed computing environment for organizations to enhance performance of computationally intensive tasks, along the lines of the SETI@home model. UD's Grid MP platform does not conform to a peer-to-peer design, as there is a centralized service provider that controls job execution.

1.a Grid MP Architecture

Each node, or device, in the Grid runs an MP Agent. A central server controls the operation of the Grid. One of its responsibilities is scheduling tasks for execution on each MP Agent. The user can submit jobs to the MP platform using various platform interfaces, e.g. MP Grid Services Interface (MGSI) and the Management Console.

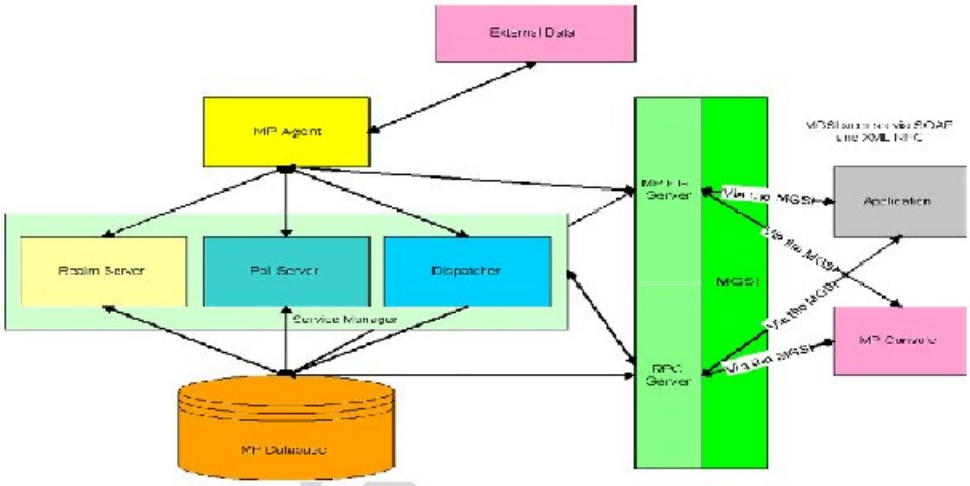

Figure 1. The Grid MP platform architecture, components, and data flow.

(Source : UD Application Developer's Guide[1])

MGSI is a web-based programming interface based on XML and SOAP. MGSI contains functions for manipulating all objects in the Grid MP platform. MGSI is used to write application services, which wrap the user's intended application for execution with preprocessing and post-processing tasks.

UD' SDK provides the capability to submit batch, data parallel, and MPI-based jobs. One advantage of using these facilities is jobs are bound to a specific resource or device only at dispatch time, instead of job submission time in traditional workload management systems.

1b. Object Model

The Grid MP platform contains a well defined object model (see Figure below). An application is a set of related programs that perform a common computational goal. Program versions enable iterative releases. Program modules contain the executable code that is platform specific.

A job is defined as an instance of the application and may consist of one or more job steps. Each job step specifies an independent program with an input data set. Each program contains one or more work units, where each work unit specifies one piece of the input data set. A work unit is the smallest object that can be scheduled in the Grid MP Platform. Data sets define a set of data files, and can belong to a job or job step; they are global. Data describes a partition of the original data set and is uniquely identified with the data set. Work units link data to a job step

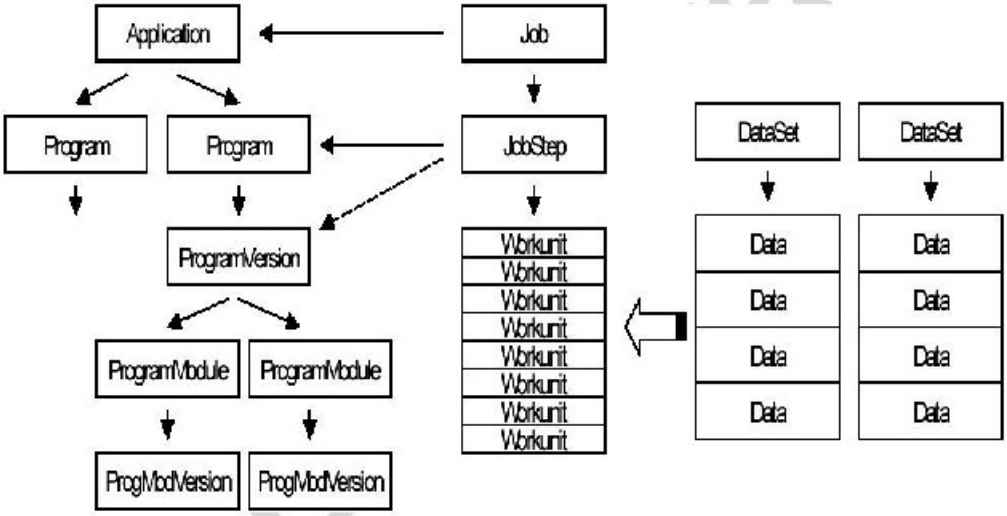

Figure 4. The Grid MP platform object model

(Source : UD's SDK)

1c. Pre and Post Processing

Before an application can be deployed on the grid certain preprocessing steps have to be carried out. This is done via the MGSI. These steps are as follows :

1 Partition data – As will be seen, we disregarded this step in our experiment. We instead sent the DNA sequences constituting out initial data set to each device, which fit the implementation we worked with.
2 Package Data Partitions
3 Log into MGSI
4 Create Job
5 Create Job step
6 Create Data Set
7 Create Data and upload Packages to the Grid MP platform File Server
8 Create work units within the job step
9 Ensure the job step is enabled to run

Post-processing is more straightforward, and essentially involves retrieval of Results/Error records, as well as any expectant output files, from the File Server. Results/Error retrieval can be done while a job step is running or after the job step has completed. This information allows construction of the "answer" to the job step. Finally, job cleanup is performed.

2 The GAML Algorithm

GAML[3] comes under a category of problems that can be called embarrassingly parallel problems. It is a genetic algorithm based on stochastic optimizations. It aims to construct an accurate Tree of Life for the world's 1.7 million species using phylogenetic analysis. Genetic algorithms are inherently parallel, but they

are not optimal, using a heuristical approach instead.

GAML starts with a set of DNA sequences. From this set it generates a tree, which are termed individuals, using mutation and reproduction of the sequences. A group of individuals constitute a population. A generation is a single iteration of the algorithm. In each iteration the individuals of a population are scored (this step is computation intensive and consumes most of the algorithm's computation). After they are scored reproduction is carried out. This step involves copying high scoring solutions into what will become the next generation of solutions. Mutation (making small changes to copied individuals) is carried out on a certain small number of the individuals in the new generation.

In serial implementations these iterations are carried out repeatedly until a satisfactory score is achieved. There are two approaches to parallelization. The first approach seeks to parallels the process of determining whether a set of trees is better than another. This approach is not suited for distributed computing environments because the overhead of sending parts of trees to each node is very high. Instead, several populations are started in parallel, increasing the chances of finding a better solution quicker. This is slightly refined by having populations share results and steer each other through better directions in the tree space. The sharing of results is done through a master node.

There are four variants on parallelization of the algorithm :

1. Full Duplex Exchange : The entire population at each node is sent to one master population. This would then select random individuals from its own population and exchange it with the individuals from the sender.
2. Alphamale Replication : The idea of master population is used again. At regular intervals the remote populations send/receive the best individual to/from the master.
3. Shielded Migrants : In this method the master population is influenced by the remote's best individuals. At regular intervals the remote sends a fixed number of individuals to the master. The master assimilates them into its population by inserting them into special slots termed “shields”. The shields cannot be altered; they can only influence the master's non-shielded individuals during replication and mutation.
4. Hybrid : A combination of alphamale replication and shielded migrants methods. Some individuals of the population communicate with the master using the former , while some using the latter.

3 Experiment

3a Approach

While there are existing parallel/MPI-based implementations of the GAML algorithm, one was not used. A suitable implementation provided with support was not available, hence a serial implementation was settled on, despite being inefficient.

The serial implementation [2] uses a configuration file that points to a data set containing DNA sequences of various organisms, along with an optional starting tree which was generated by methods other than likelihood and acts as a seed to the algorithm. The configuration file also specifies that each generated tree is to serve as the starting condition for the next generation’s calculation, and that execution stops after 10000 generations. These three files were to be bundled as two data packages (the initial tree was packaged separate) and uploaded to the grid file server via the MGSI, invoked

from a Perl script. For evaluational purposes, the application and program were to be created from the Management Console rather than MGSI interfaces.

The SDK providing MGSI services, as well as the Grid MP platform, were Version 4.0. Development was done on an i386 PC with Redhat 7.2, as recommended by the SDK Application Developer's guide.

The device group targeted was a cluster of 3 PCs also running Redhat 7.0 The application server and device group are hosted and maintained by the Texas Advanced Computing Center (TACC). As of this writing hosts had not been selected and had Linux MP Agents installed, so no hardware specification can be given.

3b Results

The immediate problem encountered with the Grid MP platform and SDK is the lack of support for UNIX environments, in this experiment's case Linux. It is quite apparent that UD developed their product and services with Windows platforms in mind. While UD does claim Linux is a supported OS, the SDK and associated documentation are very sparse. This makes the various incompatibilities with shipped binaries (e.g. MP Agent) all the more frustrating for the end developer.

The Application Developer's Guide[1] claims support on Redhat 7.2 systems with glibc 2.3 However, glibc 2.3 only began distribution with Redhat 8.0 Prior versions had glibc 2.2.x Upgrading is not a straightforward option as glibc is an essential library for most OS functions. Expecting a device group administrator to perform such a sensitive upgrade to enable MP Agents to run is not acceptable. Attempts to rebuild dynamic libraries included with the SDK against pre-glibc 2.3 levels were met with failure as other similar failed dependencies were encountered during link time. None of this is documented in SDK documentation.

An interesting, if unexplained requirement, is that the owner of a device group is responsible for MP Agent installation. She's also responsible for creation of an MP Agent user to enforce access control on the agent. The MP Agent then uses these credentials for authentication with the application server. The application server can push updates to the MP Agents it's connected to, yet it can not do a base install.

Another gating usage problem is poor documentation of the Management Console, which provides a grid-wide view of all MGSI objects. There is no usage information, only context-sensitive help, which is brief at best. This console seems quit underdeveloped in respect to addressing performance metric observation of jobs in the grid. Currently, in Version 4.0, CPU utilization of MP Agents to process workunits always shows up as 0. The MP Agent sends "pings" to the server every 5 minutes. This "ping" encapsulates a request for newly scheduled workunits and agent updates, as well as give CPU and memory usage of the local host during workunit processing. Unfortunately this information is kept in the grid database and never exposed to the user. This also exposes the lack of information on the host's load during workunit processing, which gives much contextual insight on resource usage. As for communication bandwidth and throughput, the console only exposes when a workunit was sent to an MP Agent, but not if the associated data was. There could be global data or data persistent across workunits that are still cached by the agent, hence redundant copies were not pushed out by the server.

While SDK documentation clearly defines all the MGSI objects, no design

patterns or suggestions are ever given on how to use them. An analogy would be for an object-oriented programming guide to define access modifiers (ex.. private, public) but not to describe their use. For the novice developer, the task of porting an existing application to run on the Grid MP platform is confusing, as there are many way to divide an application into multiple programs, jobs, or job steps. Which is the most efficient, i.e. Makes maximal use of grid resources, minimizing unneeded latencies?

4 Conclusion

The aforementioned problems in Linux environments are being addressed by UD through the TACC. Unfortunately the lack of exposure of resource usage by grid devices by the Management Console still exists. The simple solution to combat this is for the application developer and/or grid user to deploy user-defined performance tools with program modules.

Once the serial implementation is successfully deployed on the Grid MP platform, the more interesting question of MPI-based implementations should be explored. Prior experiences[2] suggest adequate support was not there for efficient execution with Version 3.2

References

[1]UD's SDK Documentation

[2]Derrick Zwickl

[3]**G**enetic Algorithms and Parallel Processing in Maximum Likelihood Phylogeny Interface - by Brauer, Zwickl, Hillis & others

[4]Chistopher J. Bottaro "Parallelization of the GAML Algorithm" (Project Report, Computer Sciences Department, December 2003)