A 90 Minute *SAGA* Hands-On Tutorial

ISSGC, Nice, France, July 5-17 2009

May 30, 2009

Scope of this Tutorial

The scope of this tutorial is to provide the audience with the required resources and technical knowledge (hands-on experience) to start hacking their own distributed applications with SAGA.

Prerequisites

This tutorial requires basic knowledge of the C/C++ programming language. Experience using the command line on a Linux/UNIX based operating system and a basic idea of what a compiler, a linker and a Makefile is might come in handy.

Unless this tutorial is going to be preceded by a SAGA installation tutorial, the students are required to have a fully working installation of SAGA on their laptops/lab machines (preferred), or remote access (e.g. via SSH) to a machine with SAGA installed.

Cheat Sheat

At the beginning of the tutorial we're going to hand out the SAGA Cheat Sheet (yet to be created). The cheat sheet is a double-sided (laminated, so people won't throw it away!) letter-sized piece of paper with hints and tips for writing, compiling, and running SAGA applications.

Unit I (15 min)

A minimal SAGA application. Every application that uses just a single SAGA call is considered a SAGA applications, since it triggers the whole stack. SAGA is not a framework. It doesnt impose a specific programming model or way to use it (like, e.g. MPI). Look at it as a TOOLBOX for distributed computing.

<CODE>

Unit II (15 min)

Compiling and linking. Demonstrate how to include the SAGA Makefiles to compile and link a SAGA application. Explain the concept of packages as independent shared libraries. Demonstrate static vs. dynamic linking.

<CODE>

Unit III (15 min)

Running a SAGA application. Explain what needs to be in the loader path. Explain the effects of SAGA_VERBOSE and how it can be used to debug a saga application. Demonstrate how the engine launches adaptor loading, etc... in the background.

<CODE>

Unit IV (20 min)

Hello distributed world! Submit three jobs to three machines. One returns Hello, one returns Distributed and one returns World. They may or may not return in the right order. This should give the student an idea how they could potentially speed up their application using multiple resources.

<CODE>

Unit V (20 min)

Applications The aim of this section is to see how SAGA is used to implement common *higher-level* functionality that is used by Distributed Applications (DA). Specifically, we will look at two commonly occuring functionality required by DA.

Example 1: Here we will demonstrate the ability to checkpoint, find a resource, self-migrate and restart on a different computational resource. There are multiple reasons this might be required (see lecture notes). Here we will demonstrate this capability using the **Hello distributed world!** example discussed in Unit IV. Instead of launching three jobs on three machines, we will launch one job on one machine, which will then launch itself on another machine, which in turn will do so onto yet another machine.

<CODE>

Once developed, this capability can be utilized by a wide range of different applications. In other words this capability described/shown above is independent of any specific application logic. Do you know of a (Scientific) application that could utilize this feature?

Example 2: Here we will briefly discuss MapReduce – a computational pattern made famous by Google's use for its Search Engine Infrastructure. The fundamental idea is that there is a Master which coordinates the distribution of work to a large number of Workers, and manages the merging of the output of the computation that the Workers produce. In addition to performance, a fundamental challenge is the need to be able to coordinate Master-Workers across a wide range of distributed systems.

<CODE>

Not surprisingly the code snippet above is independent of any application specific details and focusses on the assignment of workloads to workers, execution and then retrival. This specific approach adopted here relies heavily on the use of the Advert Service. Although introduced in the context of MapReduce, the requirement of coordination of distributed, often heteregenous units is a fundamental *vector* of distributed applications. (See Lecture Notes).

Excercise (20 min)

Show what you've learned! Have the students write a distributed application We have to come up with something neat, fun!!!, and simple enough that can be solved by all students within 20 minutes and the help of the cheat sheet.

<CODE>

Conclusion