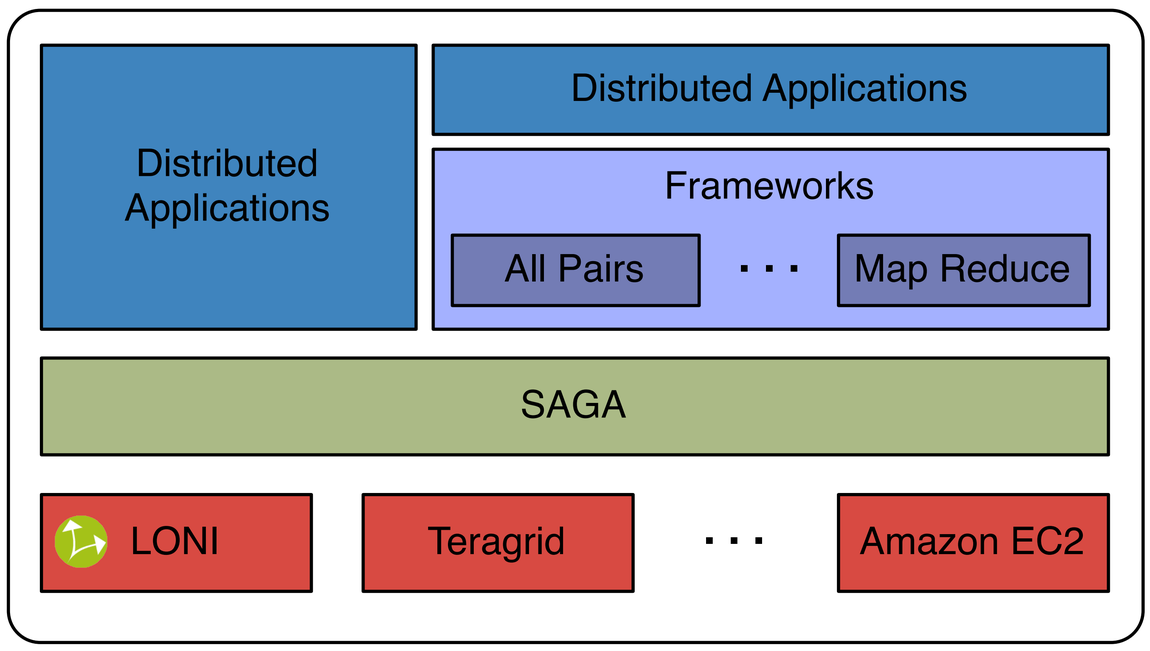
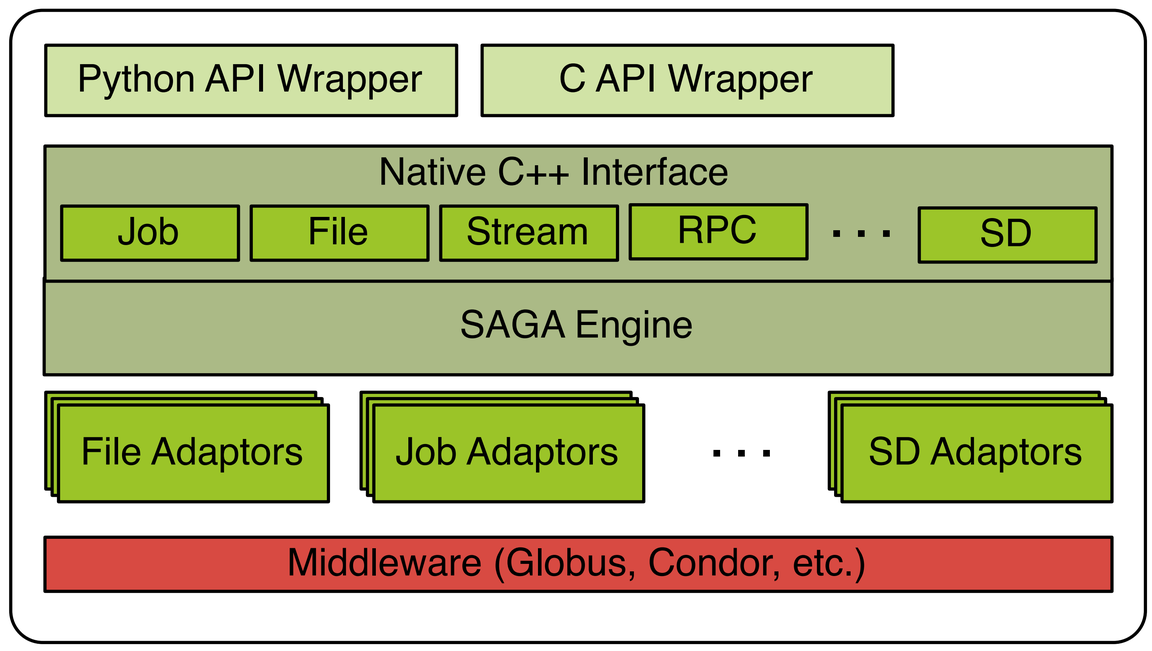
Enabling Distributed Applications with SAGA

Maybe a Catchy Subtitle ?!?

João Abecasis, Ole Weidner
Center for Computation & Technology, Louisiana State University, Baton Rouge, U.S.A.

CENTER FOR COMPUTATION
& TECHNOLOGY





Introduction

Although task farming is common practice on cluster systems using shared filesystems and local batch queueing systems, task farming on distributed Grid resources is still far from being ubiquitous due to the heterogeneous, potentially unreliable, and unpredictable nature of computational Grids. Additionally, the vast amount of middleware, transport protocols, and programming interfaces makes it exceptionally difficult to run task farming jobs transparently on distributed Grid resources.

Given the task to run a large parameter survey for a sequential code that simulates the biology and behaviour of brown shrimp in a realistic marsh environment [1] we evaluated several task farming tools including Condor-G, Nimrod/G, and others. Since none of them could completely satisfied our requirements we decided to design and implement our own middleware independent ad-hoc task farming application based on the following design objectives:

Implementation

Our task farming application [2] was designed as a set of independent agents around a central persistent Advert DB. The database acts as both job data storage and communication hub for the agents. No agent-to-agent communication is necessary. A set of command line utilities (`submit`, `control`, and `monitor`) are availabe for submitting and interacting with an active task farming instance. The basic application flow and component interaction is as follows:

1. The `submit` tool parses the TFDL job description, checks its validity, and prepares a new session.
2. The `submit` tool verifies the existence of application binaries and input data files described in (1) which can be both physical and logical URLs.
3. If steps (1), (2) were successful, the `submit` tool creates a new session structure in the Advert DB and populates it with data verified in the previous steps.
4. Once the DB is set up, the `submit` tool launches the `agents` on a set of remote hosts and exits providing a UUID for client connections (A), (B).
5. The `agents` perform a token-based leadership election to select a master agent. The master agent keeps track of the remaining agents and restores consistency in case an agent dies. If the master dies, the remaining agents perform a re-election.
6. A `agent` gathers system, performance, and queue statistics and advertises them periodically to the DB as a foundation for overall scheduling decisions.
7. An `agent` decides based on the scheduling data in the DB wether or not to start a new task. If so, it tries to retrieve an application image and input files for the next available task and runs them on its locally accessible resources (this can be a local CPU as well as a batch queuing system).
8. After a successful execution, each `agent` announces the results and the location of output and error files to the advert DB and continues with step (6).

All components were implemented in C++ using the SAGA (Simple API for Grid Applications) libraries and adaptors [3] to gain transparent access to the Grid middleware used in our testbeds which include Globus, PBS, Condor and the PostgreSQL-based Advert DB. We use SAGA's file package API for data transfer, the replica package for logical file handling, the job package for remote execution and monitoring, and the advert package API for accessing the Advert DB.

Job Description

The current version of our application uses an extended version of the JSDL 1.0 [4] XML schema - TFDL (Task Farming Description Language) to describe a task farming instance. TFDL's extension to JSDL comprises a JSDL-compliant schema add-on which allows the description of independent work packages ("chunks") for a task farming job which is currently not possible using plain JSDL. For future versions of our application, we plan to adopt the upcoming JSDL 2.0 standard which will provide all necessary schema properties for task farming job description and doesn't need any extensions.

Results

(1) We wanted to develop an easy to use task farming application capable of executing large parameter surveys on heterogeneous Grid computing resources. Using our application, we successfully conducted an initial survey of 250 input parameter permutations for the shrimp model with minimal preparation time on a Globus/PBS/Condor testbed.

(2) We wanted to show that the emerging standards in Grid computing like SAGA and JSDL finally enable developers to rapidly implement applications with focus on the application logic and the engineering process without the need to deal with middleware layer implementation details. We found that the SAGA API provides the required level of abstraction and all necessary functional packages to implement a Grid-enabled and portable application within a short timeframe (the first running prototype was ready after only one week of development). The major drawback was that the availability of middleware bindings is currently restricted to local resources (local filesystem, fork) and Globus (GRAM2, GridFTP, RLS). Especially the absence of apropriate adaptors for the widespread queueing systems PBS and Condor and a usable Service Discovery API forced us to implement parts of the scheduling and local job execution code using vendor-specific APIs and custom scripts.

However, we're looking forward to test and use the already announced SAGA Service Discovery API and batch system adaptors to leverage our application as a truly middleware independent and portable task farming tool.

References

[1] Haas, H. L., K. A. Rose, B. Fry, T. J. Minello, and L. P. Rozas, "Brown shrimp on the edge: linking habitat to survival using an individual-based simulation model", 2004.

[2] SAGA-based task farming application, [Online]. http://cct.lsu.edu/~oweidner/tf/

[3] SAGA - A Simple API for Grid Applications, [Online]. http://saga.cct.lsu.edu

[4] JSDL Spec., Version 1.0 [Online]. http://www.ogf.org/documents/GFD.56.pdf