Overview of EnsembleMD Toolkit

Nikhil Shenoy
Electrical and
Computer Engineering
Rutgers University - New Brunswick
Piscataway, NJ 08854
Email: nrs76@scarletmail.rutgers.edu

Shantenu Jha
RADICAL Cybertools
Piscataway, NJ
Email: shantenu.jha@rutgers.edu

***Abstract*—Much of today's scientific experiments have become computation driven, and the amount of data that needs to be processed can be very large. Traditional data processing techniques relied on scheduling tasks directly through the job scheduler on a cluster, and then running the job using a Message Passing library to achieve the desired parallelism. However, many drawbacks were discovered with this approach, with the most prominent being extended wait times for jobs in the scheduling queue. This problem, coupled with scientists' increasing demand for fast, responsive simulations, has led to inefficient systems and a lack of progress.**

This research seeks to eliminate the drawbacks of traditional processing methods by utilizing an existing Pilot-Job system, RADICAL-Pilot, to process problems in molecular dynamics. The resultant software, the EnsembleMD Toolkit, is designed to to efficiently manage and assign tasks to distributed resources using Pilots, and to abstract the details of resource management away from the user. Such an abstraction will allow the user to focus on the science of his simulation rather than the complexities of running the simulation. Specifically, this work analyzes the performance of EnsembleMD modules and details the development of additional modules for the API.

I. Introduction

Traditional distributed systems utilize batch queueing systems in order to schedule jobs to the system's resources. In such systems, scripts are written to execute several different tasks sequentially, and a scheduler assigns the tasks the resources it requires. The problem with this approach is that since a task will most likely require only a fraction of the available resources, the remainder remains idle. Leaving cores unutilized severely limits the efficiency and throughput of the system and must be avoided. Additionally, the scheduler does not account for attributes specific to the application, which include the number of cores, the size of the task, and whether the task has multi-threaded, multi-process, or serial execution. These problems are carried with the user who, in many cases, is attempting to run computation-heavy scientific simulations.

To alleviate this problem, the concept of Pilot-Jobs was introduced.

A pilot job is a special type of container designed to process and manage several different tasks during its lifetime. In traditional processing schemes, a job generally consists of running the same executable repeatedly with varying parameters, with each instance becoming a process that is submitted to the scheduler. Each of these instances must wait to have their resource requests fulfilled and then be correctly scheduled in order to execute. In a pilot scheme, information about each instance is associated with the pilot, and only the pilot is placed into the scheduling queue. Once the scheduler schedules the pilot, it then establishes communication with the user and accepts tasks. The user can now schedule tasks directly to the pilot, which then runs the task using a portion of its allocated resources, assuming that the task's resource requirement does not exceed the amount of available resources. If resources are still available after the scheduling of the initial task, the pilot will accept additional tasks and schedule them to those resources. The user can continue to schedule tasks to the pilot until the wall time of the pilot expires.

The main advantage in using a pilot system is that one can circumvent the scheduler when running multiple tasks. By placing only the pilot into the scheduling queue, the user obviates the time necessary to request and assign resources for the total number of tasks. This can result in a decreased time-to-completion for each task, and high throughput of the number of tasks. More importantly, such a scheme leads to efficient usage of resources, as not only can a single pilot keep its local resources occupied for long periods of time, but a series of pilots can ensures that a large percentage of the entire grid's resources are kept busy. Since the submission of the pilot and its workload is decoupled from the assignment of resources, complex applications can be written to take advantage of all the resources in the system.

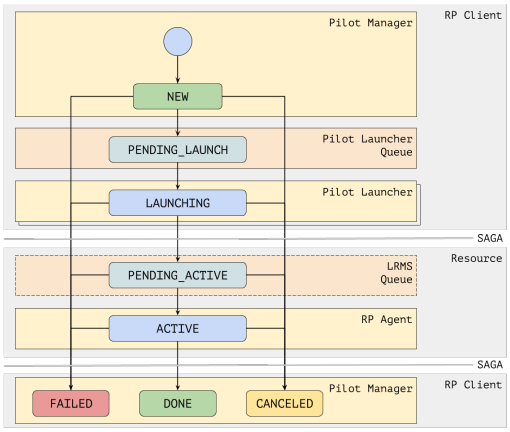
A. RADICAL-Pilot

RADICAL-Pilot (RP) is an implementation of a pilot-job system created by the RADICAL-Cybertools research group, and is utilized by the EnsembleMD Toolkit. RP consists of three main components; the Compute Unit, the Pilot, and the Agent. These components are abstractions for the task of a workload, the resource placeholder, and component of a pilot that manages the execution of units. The Compute Unit and the Pilot are designed as stateful entities, and the Agent is responsible for shepherding them through the different states.

At any given time, the Pilot is in one of the following states: New, Pending_Launch, Launching, Pending_Active, Active, Done, Cancelled, or Failed. Naturally, a Pilot begins in the New state when it is created. Once the Pilot has been instantiated, internal modules change the Pilot's state

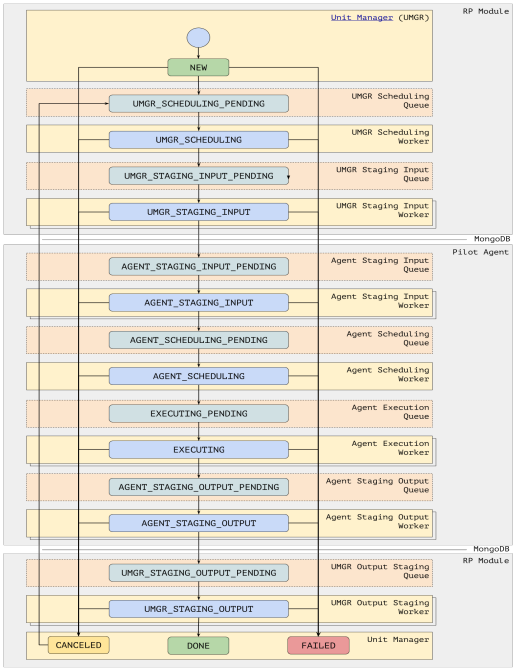
to Pending_Launch, and after being placed in the queue, to Launching. The scheduler will then schedule the Pilot as it would any other normal job, and the Pilot's state is changed to Pending_Active. After the Pilot has been granted its resources, it starts the Agent and changes its status to Active. In this state, the Pilot will continue to accept tasks from the application and schedule them efficiently across the given resources until no more tasks remain. At this point, the Pilot can be in either Done, Cancelled, or Failed. Clearly, the Pilot will be Done if all tasks have executed without any errors. If the Agent is interrupted, such as with the interrupt signal, then the Pilot will become Cancelled. If an error occurs, the Agent changes the Pilot state to Failed and stops execution.

Fig. 1. Pilot State Diagram



Compute Units have a similar set of states. A Compute Unit can take on one of the following states: New, UMGR_Scheduling, UMGR_Staging_Input, Agent_Staging_Input, Agent_Scheduling, Executing, Agent_Staging_Output, UMGR_Staging_Output, Done, Cancelled, Failed. The UMGR in the state names stands for the Unit Manager which, true to its name, manages units and what Pilots they are associated with. This occurs in the UMGR_Scheduling phase, after the Compute Unit is instantiated in the New state. In the UMGR_Staging_Input phase, the Compute Unit's state transitions differ from that of the Pilot; since the Compute Unit represents the task to be executed, it must receive all the input it needs in order to run. During this phase, the Compute Unit receives the name of the executable, command line parameters, input datafiles, and any other information. The Agent ensures that the input staging occurs properly in the Agent_Staging phase and begins executing the task on a portion of the resources in the Executing state. Once the task finishes executing, the Agent_Staging_Output and UMGR_Staging_Output states gather any output data and returns it to the application. Finally, the Compute Unit enters one of the three final states in the same way that the Pilot does.

Fig. 2. Compute Unit State Diagram



B. *EnsembleMD Toolkit*

The molecular dynamics community works on simulating the interactions of atoms and molecules, which can be generally classified as N-body problems. To run these simulations, the scientists must submit their tasks to distributed computing resources (DCRs) using software specifically designed for their experiments. There are many such softwares throughout the community, but since there is no standard that integrates all the functionalities into a single framework, progress is hindered due to a lack of portability and extensibility. The absence of this standard prevents scientists from adapting to and creating new simulation patterns, leading to the larger problem of unscalable software.

Heterogeniety of DCRs compound this problem. Execution of a task on a DCR requires an intimate knowledge of all available resources; each instance of the existing molecular dynamics software is designed for at most a handful of resources, and the architectures of all of those systems can differ. Introducing a new DCR into the software could require a scientist to restructure major components, inevitably leading to additional bugs, and would cost the scientist time and funding for the project.

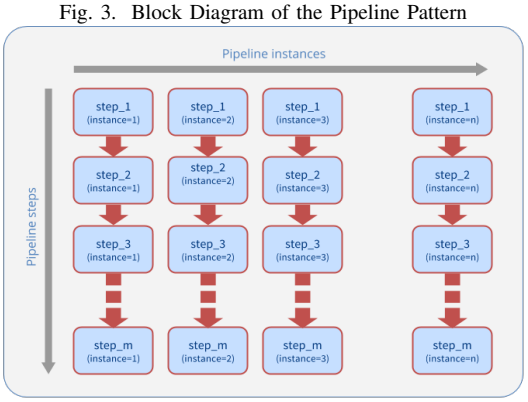
In this paper, we present the EnsembleMD Toolkit as a solution to these problems. The Toolkit is centered around the concept of an ensemble, which is defined as task consisting of several simulations with different input parameters. Using the ensemble as the primary entity, our software provides three unique features:

(i) Instantiation of ensembles is separated from the simulation tool
(ii) Execution patterns based on ensembles

(iii) Decoupling of "what to do" from "how to do it"
(iv) Dynamic execution on a range of DCRs using the RADICAL-Pilot API

The EnsembleMD Toolkit consists of three main components; the Application Pattern, the Execution Context, and the Kernel Plugin. The first component that the user interacts with is the Application Pattern, which is a general template for executing tasks. For example, the Pipeline Application Pattern can run a collection of tasks sequentially as a series of steps. In this scheme, the pattern will wait for each step to finish before moving on to the next step. The pattern also allows for several instances of a given pipeline, as shown in Figure 1, if redundancy is required. Other available patterns include Simulation-Analysis and the Replica Exchange.

Fig. 3. Block Diagram of the Pipeline Pattern

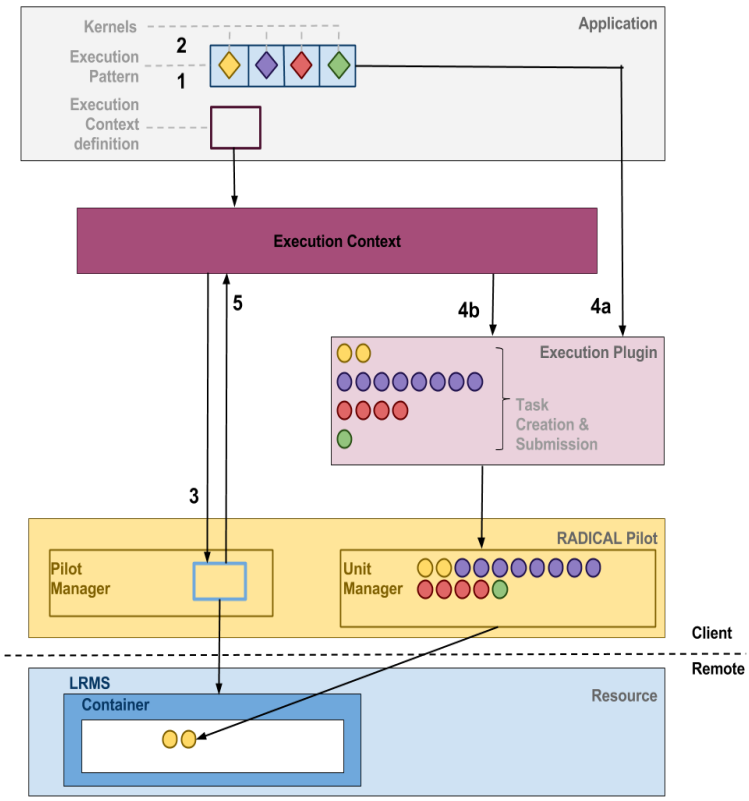


These patterns present the user with high-level descriptions of control flow which are task-independent. The user need only choose a pattern and provide it with the details necessary for the simulation, and the API takes care of the rest. By doing this, the details of executing the simulation are abstracted from the user, providing an interface that makes translating biological experiments into computer simulations very simple.

The Execution Context represents the DCR that the simulation is running on. It contains the infrastructure needed to interface with the machine itself, and is the agent that allows the Toolkit to adapt to and include a variety of resources. Its main functions are to allocate and deallocate the resource, as well as run an execution pattern on the resource. Aside from specifying which resource to use, the user does not interact with this component directly as it is meant to be behind the wall of abstraction.

Finally, the Kernel Plugin represents the scientific tool to be used. This can be anything from molecular dynamics executables such as Amber to simple tasks such as counting the number of characters in a file. The Toolkit provides a list of built-in kernels, but provides a mechanism for users to add their own. The Kernel Plugin abstracts aspects of the tools that are specific to a particular resource and presents a uniform interface with the user can define tasks.

Fig. 4. The EnsembleMD Toolkit Architecture



II. Background

III. Previous Work?

IV. Experiments

A. Performance

1) Weak Scaling:

2) Strong Scaling:

B. New Modules

V. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.