Performance Summary of Ensemble-MD Patterns

Nikhil Shenoy

April 14, 2016

Abstract

Modern software, no matter its application, always incorporates performance as a design factor. Without efficient implementations, the software becomes less attractive to the user because it does not accomplish the user's tasks in a timely fashion. Such factors are seriously considered when designing software for scientific computing, as many simulations in areas such as molecular dynamics require efficient tools to process the ever-increasing amount of data. In this article, we discuss the EnsembleMD-Toolkit and benchmark the efficiency of two standard Patterns using a sample workload. We assume that the reader has a basic familiarity with the Toolkit, and we will define only parameters relevant to the performed experiments.

1 Introduction

Traditional distributed systems utilize batch queueing systems in order to schedule jobs to a high performance machine's resources. In such systems, scripts are written to execute several different tasks sequentially, and a scheduler assigns the tasks to 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 remaining resources will stay idle. Leaving cores unutilized severely limits the efficiency and throughput of the system and must be avoided. Additionally, the total waiting time in the queue for a simulation can scale quickly in the number of component tasks, which can significantly increase the time to completion. These problems burden the user who, in many cases, is attempting to run computation-heavy scientific simulations [1].

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 a pilot scheme, information about each task of a simulation is associated with the pilot, and only the pilot is placed into the scheduling queue. Once the scheduler schedules the pilot, it instantiates an agent that pulls information about the tasks to execute from the MongoDB. Provided that enough resources are available, those tasks are then executed. 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 assigned resources occupied for long periods of time, but a series of pilots can keep a large portion of the entire grid's resources 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 [2].

Based on these advantages, the RADICAL-Cybertools group developed RADICAL-Pilot, which is a Python API that provides user-friendly abstractions for the pilot model. By abstracting the pilot paradigm, one can use RADICAL-Pilot as a foundation for more complex use-cases that require high performance machines. The EnsembleMD-Toolkit, an API designed for large scale molecular dynamics simulations, is a prime example of this, as it utilizes RADICAL-Pilot to implement simulations that are popular within the molecular dynamics community.

1.1 Ensemble-MD Toolkit

In this article, we present the EnsembleMD Toolkit (EnMD) as one of the complex applications built on top of the RADICAL-Pilot architecture, and provide a brief overview of the software here. The execution flow of the Toolkit is centered around three main components; the Application Pattern, the Execution Context, and the Kernel Plugin.

The first component that the user interacts with, the Application Pattern, is a general template for executing tasks. 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. 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 [3], [4].

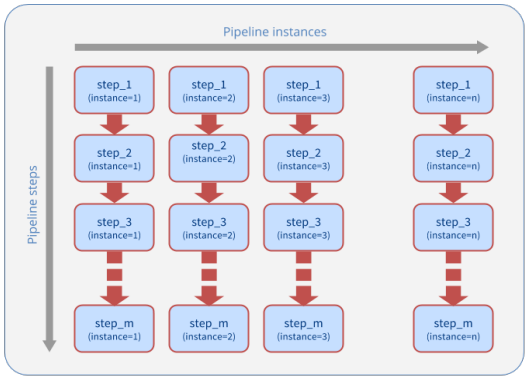

Figure 1: Block Diagram of the Pipeline Pattern [5]

The Execution Context represents the distributed computing resource 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 [3].

Using this three basic components, EnsembleMD is able to characterize biological phenomena and run simulations specific to those phenomena. However, recent efforts in the development of the Toolkit have led to applications in astronomy, specifically in monitoring the *movement of galaxies* (not sure about this bit). The abstractions for biological phenomena provided by the Toolkit are thus shown to not only be effective for those types of simulations, but can also be extended to fit other science applications as well.

2 Experiments

The performance experiments described in this section were designed to analyze the Pipeline and Simulation Analysis Patterns. We define the experiment configuration key equations, and explain useful diagrams here.

The experiment is comprised of a two-stage workload; the first stage consists of creating a 10Mb file of random characters, and the second stage performs a character count on this file. In Table 1, we have included the commands used to execute each stage as well as the average execution time

from the Bash Shell. We also use the environment parameters specified in Table 2.

Kernel	Bash Command	Avg. Execution (seconds)
misc.mkfile	base64 /dev/urandom | head -c 100 > test.txt	.008
misc.ccount	grep -o . test.txt | sort | uniq > out.txt	.004

Table 1: Kernels, their commands, and their expected execution times.

Parameter	Value
EnsembleMD-Toolkit version	0.3.14-27-g65bc062
EnsembleMD Branch	devel
RADICAL-Pilot version	0.40.1
Target Machine	XSEDE Stampede

Table 2: Environment Parameters.

To examine the performance of the Toolkit, we measure the duration of several key components of the execution. These components are defined in Table 3.

Component	Definition
EnsembleMD Core Overhead	Overhead incurred by EnsembleMD Toolkit when allocating and deallocating a cluster.
Data Movement	Overhead incurred in moving input data from the local machine to remote node, and from moving the output data from the remote node to the local machine
X Execution Time	Total time required to complete phase X of the Pattern
RADICAL-Pilot Overhead	Overhead incurred by the underlying RADICAL-Pilot API

Table 3: Measured Components

We then place timestamps at appropriate places in the pilot's execution corresponding to each of these components, as shown in Figure 2.

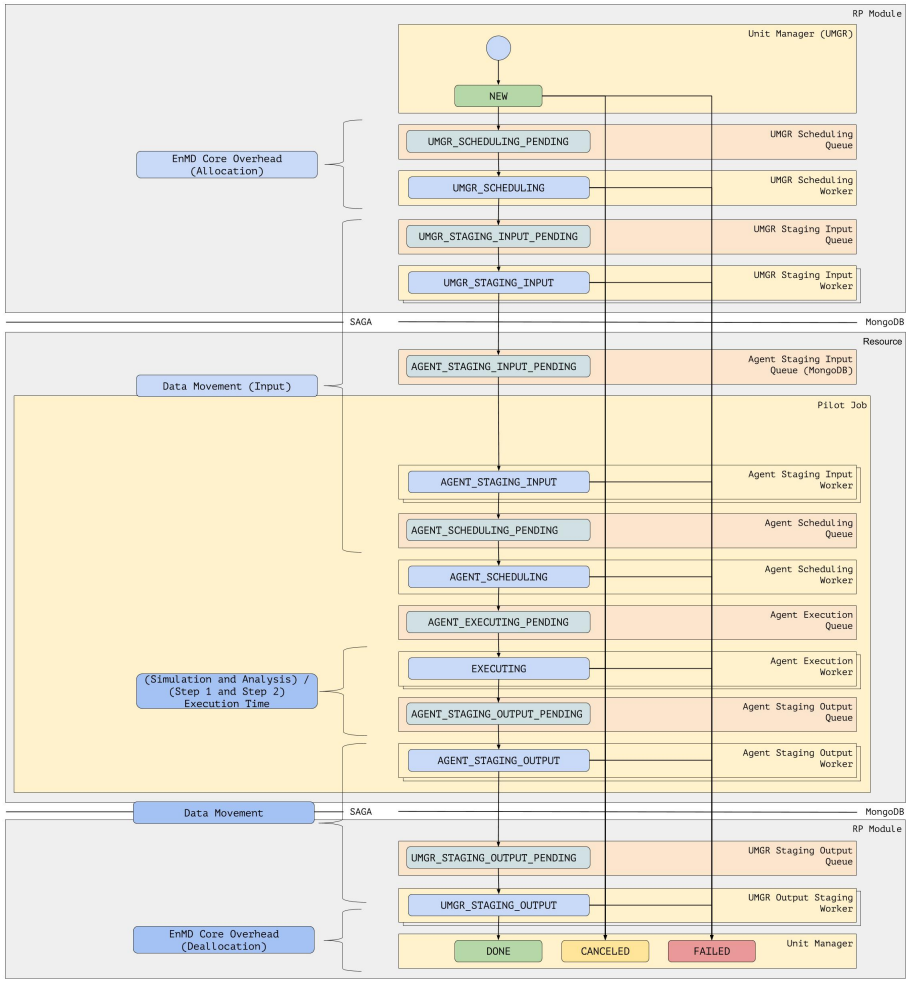

Figure 2: Mapping of parameters to pilot state model. Derived from [6].

Each of the components exhibited in Figure 2 is governed differences in timestamps. The following equations explicitly describe those differences.

$$EnMDCoreOverhead = (alloc\_stop - alloc\_start) + (dealloc\_stop - dealloc\_start) \quad (1)$$

$$EnMDPatternOverhead = (step1\_wait - step1\_start) + (step1\_stop - step1\_res) + (step1\_wait - step1\_start) + \quad (2)$$

$$RADICAL-PilotOverhead = ((step1\_wait - step1\_res) - step1\_data\_movement - step1\_execution\_time) + (\quad (3)$$

$$DataMovement = ((step1\_Done - step1\_PendingAgentOutputStaging) + (step1\_Allocating - step1\_StagingIn \quad (4)$$

$$Step1ExecutionTime = PendingAgentOutputStaging - Executing \quad (5)$$

$$Step2ExecutionTime = PendingAgentOutputStaging - Executing \quad (6)$$

Figures 3 details the measurements taken within the Execution Time state shown in Figure 2. The start and stop times indicate when execution starts and ends for that particular stage. The wait time indicates when the stage submits all of its tasks and waits for them to finish. Finally, the res time is defined as the point when all tasks of that stage have finished execution and the EnMD Toolkit resumes execution to prepare for the next stage. The terms in the parenthesis, (step1/sim) and (step2/ana), do not indicate division; this notation is used to condense the Pipeline Overhead diagram and the Simulation Analysis Overhead diagram into one figure. In the Pipeline case, the diagrams would include the "step1" and "step2" terms, whereas the Simulation Analysis case would include the "sim" and "ana" terms.

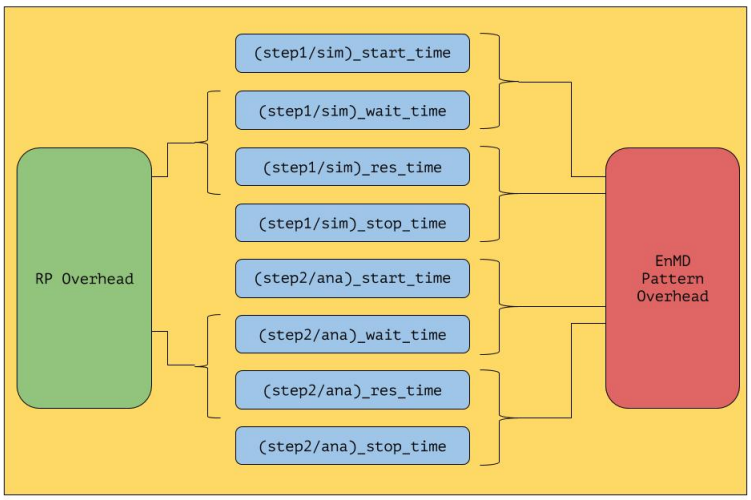

Figure 3: EnMD and RP Overheads. Derived from [6].

2.1 Types of Scaling

For these experiments, we measured performance as a function of the number of cores allocated to a script as well as the number of instances

of the Pattern being executed. We initially implemented weak scaling, in which the number of cores scales at the same rate as the number of instances, with the goal of observing a relatively constant execution time. The intuition behind this is that the core to instance ratio stays the same, which implies that the work per core/instance combination is the same. We also perform strong scaling, in which we hold the number of instances constant while we scale the number of cores. In this case, we expect that the Pattern will make use of the additional cores to finish the task more quickly. The scale that we use for these experiments is [1,16,32,64,128].

In each script, we measured the time taken to complete various stages of execution. These will be elaborated upon in the subsection for each pattern.

2.2 Pipeline

For the Pipeline pattern, we measured the parameters defined in Table 3.

Parameter	Definition
EnMD Core Overhead	(alloc_stop-alloc_start) + (dealloc_stop-dealloc_start)
EnMD Pattern Overhead	((step1_wait-step1_start) + (step1_stop-step1_res)) + (step2_wait-step2_start) + (step2_stop-step2_res))
RP Overhead	((step1_wait-step1_res) - step1_data_movement - step1_execution_time) + ((step2_wait-step2_res) - step2_data_movement - step2_execution_time)
Step 1 Execution Time	PendingAgentOutputStaging - Executing
Step 2 Execution Time	PendingAgentOutputStaging - Executing
Data Movement Time	((step1_Done - step1_PendingAgentOutputStaging) + (step1_Allocating - step1_StagingInput)) + ((step2_Done - step2_PendingAgentOutputStaging) + (step2_Allocating - step2_StagingInput))

Table 4: Pipeline Definitions

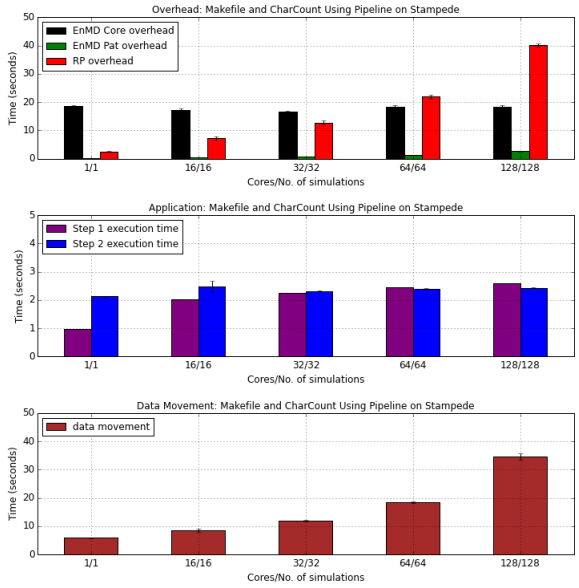

Figure 4: Weak Scaling with the Pipeline Pattern

Figure 4 shows the scaling behavior of each of the parameters. In the first graph, we observe that the EnsembleMD Core overhead stays relatively constant at about 18 seconds throughout the scaling. We also see that the EnsembleMD Pattern Overhead is at most 3 seconds, which is a fraction of the Core overhead. Finally, we see that the RADICAL-Pilot overhead increases rapidly as the scale increases. Ratios between RP overheads from a configuration to the previous configuration yield a value of about 1.8, which implies that the overhead is growing linearly. In the second graph of Figure 1, we see that the execution times for both Step 1 and Step 2, which were the Makefile and Character Count respectively, stay very much constant throughout the scaling, except for the anomaly of Step 1 in the first configuration. This is to be expected, as the settings in the configuration should have no effect on how long it takes to run the underlying Bash commands. However, the times shown are much larger than those observed when the commands are executed directly from a Bash prompt. It is possible that the extra time is due to overhead from a combination of EnsembleMD and RADICAL-Pilot, but we have not dissected this additional component. Finally, the data movement shows a steady increase in duration. The ratios from one configuration to the previous configuration fluctuate from 1.6 to 1.9, but generally imply a linear increase as a function of the number of cores and execution instances.

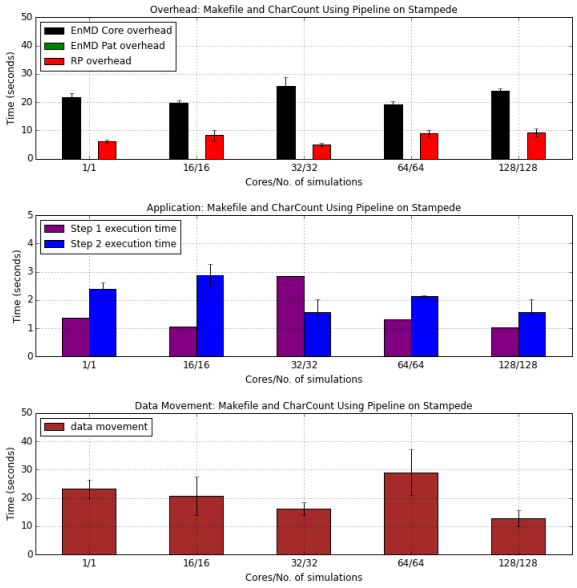

Figure 5: Strong Scaling with the Pipeline Pattern

Figure 5 displays the strong scaling results from the Pipeline. Looking at the EnsembleMD Core Overhead, we find that it averages to around 22 seconds. On the other hand, the Pattern Overhead was measured at 15ms on average, so the values at each configuration did not register on the scale of the Core Overhead. The RP overhead remains approximately constant at around 8 seconds. The execution times for Steps 1 and 2 show a much different trend than they did in the weak scaling experiments; in general, Step 1 showed execution times that were closer to the actual execution time, but were still three orders of magnitude greater. Step 2 fluctuated much more than it did during the weak scaling experiments. Data movement shows a general decline in duration, aside from the anomaly with 64 cores. This may be due to the increased number of cores available for the movement.

2.3 Simulation Analysis

For Simulation Analysis, we considered the definitions in Table 4.

Parameter	Definition
EnMD Core Overhead	(alloc_stop-alloc_start) + (dealloc_stop-dealloc_start)
EnMD Pattern Overhead	((sim_wait-sim_start) + (sim_stop-sim_res)) + ((ana_wait-ana_start) + (ana_stop-ana_res))
RP Overhead	((sim_wait-sim_res) - sim_data_movement - sim_execution_time) + ((ana_wait-ana_res) - ana_data_movement - ana_execution_time)
Simulation Execution Time	PendingAgentOutputStaging - Executing
Analysis Execution Time	PendingAgentOutputStaging - Executing
Data Movement Time	((sim_Done - sim_PendingAgentOutputStaging) + (sim_Allocating - sim_StagingInput)) + ((ana_Done - ana_PendingAgentOutputStaging) + (ana_Allocating - ana_StagingInput))

Table 5: Simulation Analysis Definitions

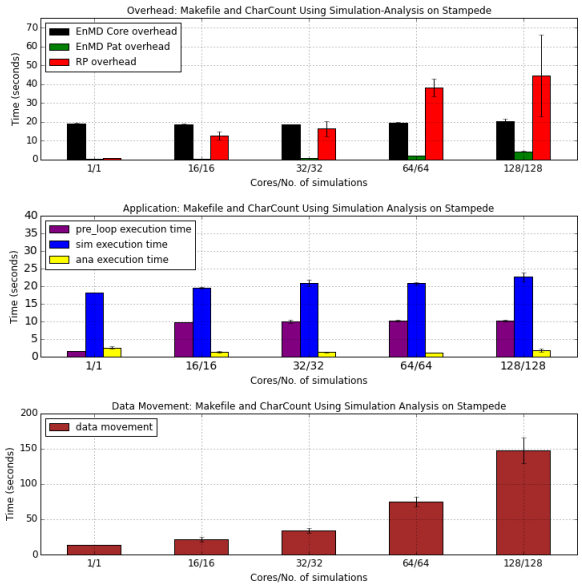

Figure 6: Weak Scaling with the Simulation Analysis Pattern

In Figure 6, we see scaling behavior similar to what we saw in the original pipeline weak scaling experiment. The EnsembleMD Core Overhead is essentially constant, the Pattern Overhead is small in comparison and is

capped at around 5 seconds, and the RP overhead does show an increase across the combinations. However, the RADICAL-Pilot overhead does not show a distinct linear progression in the same fashion as the Pipeline Weak Scaling plot did. The second plot shows the phases of execution of the experiment. The pre-loop phase contained no logic, so our inference is that the times shown are the overhead introduced to make the call to the pre-loop. The Simulation execution time, which contained the Makefile Kernel, was constant throughout the scaling, but took longer on average than it did for the Pipeline. The Character Count, encapsulated by the Analysis stage, was truer to the values recorded in the Pipeline Weak Scaling plot. The Data movement plot shows a similar increase in the time needed to download the output data, but the trend does not seem to be linear.

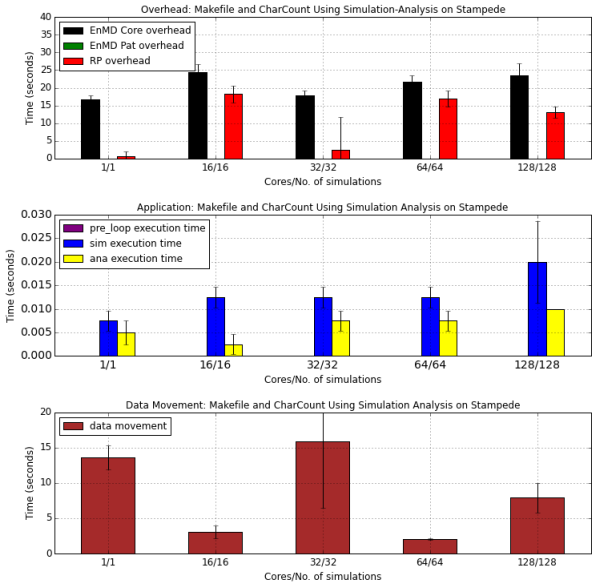

Figure 7: Strong Scaling with the Simulation Analysis Pattern

Finally, we examine the behaviors of each measurement during the Strong Scaling experiments with Simulation Analysis in Figure 7. EnsembleMD Core Overhead wavers around 20 seconds, whereas the RADICAL-Pilot Overhead has a large amount of variation in its time. Again, the EnsembleMD Pattern Overhead averages to approximately 20ms, and thus are not be visible on the same graph. In the Simulation phase, we find that the execution time has been reduced dramatically, and that a representative value for it could be 13ms. The Analysis phase also varies slightly, but can be approximated at 7ms. The Data Movement durations

are troubling, seeing as there is not a clear pattern to how the different values were obtained. The scaling does not seem to have had a visible effect on the duration of this phase.

2.4 Reproducibility

The experiments described in this paper can be reproduced easily. We enumerate the steps as follows:

1. Set up virtual environment on local machine using virtualenv tool.
2. Install the EnsembleMD Toolkit using the instructions found in the documentation (http://radicalensemblemd.readthedocs.org/en/stable/installation.html)
3. Acquire an account on the remote target machine. In this case, the target machine was XSEDE's Stampede.
4. Generate a free MongoDB instance using MongoLab (https://mlab.com/) and note its access URL.
5. Modify the bag_of_tasks.py and the simulation_analysis_loop.py examples in radical.ensemblemd/examples folder. This includes changing the kernels to Makefile and CharacterCount and changing the size of the input file to 10Mb
6. For each core count in the scale [1,16,32,64,128], run the application at least four times. This allows the experimenter to verify the accuracy of his results.
7. Set the resource to the remote target machine for which you now have an account.
8. In the case of weak scaling, set the number of instances equal to the core count. In the case of strong scaling, set the number of instances to 1, but continue to alter the core count in the same way as in the weak scaling case.
9. Use Tmux or another terminal multiplexer to start the job on your machine. You can detach the job once EnsembleMD begins executing on the remote target machine. It is recommended to start the job at night, as it will be completed by the next morning.
10. If any errors occur while testing, remember to safely close and stop the EnsembleMD script. Also make sure that the requrested resources are not still associated with one's account on the remote machine. One can verify this by logging in to that machine and checking the job status.
11. Remember to run all jobs with the following environment variables:
    - RADICAL_ENMD_VERBOSE=DEBUG
    - RADCIAL_ENMD_PROFILING=1
    - RADICAL_PILOT_DBURL= <your mongodb url>

3 Conclusion

This set of experiments were conducted to further understand the EnsembleMD Toolkit's scaling behavior. Based on the data from our experiments, we were able to identify trends that hinted at the Toolkit's capabilities for different core-task configurations. We considered stages of operation within the Toolkit that best exemplified its functionality; the Core Overhead, the Pattern Overhead, the RADICAL-Pilot Overhead, the Task Execution Time, and the Data Movement. While not all of these parameters showed distinct trends, we were able to glean information from a few of the parameters.

The Task Execution time, for both the Makefile and Character Count steps, were found to be relatively constant across most configurations. This behavior is expected, as the time required to execute a bash shell command should only change if different input is given to it. In our case, the kernels in each step are doing the exact same tasks in every configuration, so a change in the execution time would be troubling. Even though the execution time was constant, we noticed that it was much higher than the time the same tasks took when executed from the Bash command prompt. This indicates that there are additional RADICAL-Pilot and EnsembleMD wrappings around the Bash commands that increase the duration. While dissecting that overhead was not the goal of this experiment, it is a candidate for future work. Throughout all the experiments performed, we observed that the EnsembleMD Core Overhead stayed relatively constant over all configurations. This implies that users and developers can rely on the performance of the EnMD Core staying constant regardless of the tasks the user gives it.

In almost all measurements of the EnMD Pattern Overhead, we found that its duration was on the order of milliseconds. This is useful to users because they can be assured that any preparation done by EnMD to before executing the task is very efficient and is not expected to hinder the completion of the task. For developers, this indicates that the Toolkit makes quick transitions between states in the flow of execution. RP Overhead seems to scale in some cases, stay constant in other cases, and fluctuate in others. Because of this, we cannot extract a useful pattern and characterize RP's behavior at scale solely based on these experiments. As a majority of RP was implemented before EnMD, we would expect its performance at scale to have been worked on extensively and made reliable. We would need more experiments to properly pin down the behavior, as knowledge of how efficiently the pilot functions is crucial to the application's execution.

In these experiments, we measured data movement as the time needed to download the output of our tasks from the remote machine to the local machine, seeing as all the input data was being generated on the remote machine. Both of the weak scaling cases showed that the time required for the data movement was directly proportional to the core/instance configuration. The Pipeline case seemed to imply a linear relationship between the variables, whlie the Simulation-Analysis case implied a more quadratic relationship. We have not investigated the relationship in detail, but determining such relationships would allow the user to restructure his

application and explore the tradeoffs between effiency and data usage. Unfortunately, we were not able to notice distinct patterns in the strong scaling cases. We do believe that relationships similar to the ones suggested by the weak scaling experiments do exist, but further testing would be required to find them.

4 Future Work

Performance testing is an ongoing body of work in any scientific experiment, and the results presented here can be extended into any different directions. First, one could examine the relationship between the time taken to process varying sizes of data and the number of cores given to process that data. Such information would be useful in gauging EnsembleMDâĂŹs capability to handle large and small amounts of data. The same experiments could be run on all the high performance machines supported by the Toolkit, with the expectation that the user gets similar performance no matter which machine he uses. Next, one could consider the ToolkitâĂŹs role in data movement; in these experiments, we considered only the ToolkitâĂŹs involvement in transfers between the local and remote machine. An interesting avenue to explore would be the duration of all the operations and changes that the Toolkit makes to the data while it is on the remote node but not being processed. This, combined with this articleâĂŹs measurment of local/remote transfers, may better reflect the overhead incurred by data movement. Finally, discovery of best-fit functions for the important components of the Toolkit, such as the Pattern Overhead, RP Overhead, and Data Movement, would characterize the Toolkit in an easy-to-use way. The benefit of using a simple function to approximate the behavior of a Toolkit component would provide the user with instant information on how to structure his application, leading to increased usability.

5 Lessons Learned

Analyzing the performance of a system can lead to many insights about the system, and about the attitude with which one approaches experimentation in general. During my experience with testing the EnsembleMD Toolkit, I learned a variety of concepts which I believe have improved my abilities as a researcher.

One of the most important concepts I learned about is careful design of experiments. At the beginning of my experimentation, I had a tendency to dive into the data head first without a solid plan for what I was measuring. I tried to find relationships between every pair of variables in the data set, but I soon learned that this was much too inefficient to produce good results. In many of those tests, the relations I found did not yield any useful information about the system, which rendered all the time I spent running those simulations useless. Not only would these types of experiments not produce results, but adding just one variable would dramatically increase the complexity of running the entire set of

simulations. After this experiment, I learned to focus on specific behaviors of the system I wanted to examine, such as strong scaling, and tailored my experiments towards those. This resulted in concise, informative results that provided useful information for the development team.

I also understood the importance of reproducibility of experiments. Providing the parameters to reproduce an experiment allows me to go back to my experiment after some time and still expect to have the same results. Also, others can look at my work, run the same experiment, and verify whether my interpretation of the data is valid. In the greater scientific community, reproducibility allows a scientist to test the foundations on which his current experiment is based. If he thinks his experiment went wrong based on work done by others, he can go and re-run the experiment to either verify or change his assumptions. Sometimes the initial interpretation of the results was wrong, and another trial yielded a more significant, appropriate result. This allows the community to make sure that the assumptions from previous experiments are a solid foundation upon which new work can be done.

Adapting existing code to my experiments improved my skills as a developer and widened my knowledge of EnsembleMD and the rest of the RADICAL Cybertools stack. In the all the projects I had done before joining the group, I was required to implement all functionality from scratch, which augmented my understanding of class concepts but did not reflect real software development. In designing my experiments, I was forced to look at code written by others, understand it, and adapt it to my work. As I read, I understood why layers of abstraction were implemented between components of the Toolkit, and between layers of the RADICAL Cybertools stack. My view of the system changed from a collection of disjoint Pilot operations into a series of well-defined state transitions. Learning about the architecture in this way gave my experiments more context, and allowed me to remove measurements which didn't capture the interactions between important components.

Finally, I gained a greater appreciation for reading technical papers. I read a variety of papers written by RADICAL developers and other members of the community, I understood that impactful papers require careful design in order to be useful. In specific, the specific sections of a paper must have a clear focus and a proper transition into the following section. I especially noticed how important the introduction and background are in setting the context for the experiment. If carefully crafted, then the user will be more willing to read the rest of the paper and understand why one's work is so important. The sections describing the experiment must be detailed, yet concise. It is best to include only enough information to reproduce the experiment and explain why the work is being done; any more information could cause the reader to become confused and lose sight of the experiment's purpose. Finally, the analysis and conclusion should bridge the context laid out in the opening sections and the results from the experiment to summarize the impact of the work. While many essays and other pieces of writing do a have a similar structure, I was able to learn how the scientific community adapted that structure to fit its needs and present its work to the world. Going forward, I can make use of this in order to make my own contributions.

References

[1] V. Shah, A. Treikalis, and S. Jha, "Towards effective selection of collective xsede resources," 2012.

[2] A. Merzky, M. Santcroos, M. Turilli, and S. Jha, "Radical-pilot: Scalable execution of heterogeneous and dynamic workloads on supercomputers," 2015.

[3] V. Balasubramanian, A. Treikalis, O. Weidner, and S. Jha, "Ensemble toolkit: Scalable and flexible execution of ensembles of tasks," 2016.

[4] A. Treikalis, A. Merzky, H. Chen, T. Lee, D. York, and S. Jha, "Repex: A flexible framework for scalable replica exchange molecular dynamics simulations," 2016.

[5] RADICAL-Cybertools, "Pipeline pattern." http://radicalensemblemd.readthedocs.org/en/stable/_images/pipeline_pattern.png, 2016.

[6] RADICAL-Cybertools, "Radical-pilot global state model." https://radicalpilot.readthedocs.org/en/stable/_images/global-state-model-plain.png, 2016.

[7] M. Santcroos, S. Olabarriaga, D. Katz, and S. Jha, "Pilot abstractions for compute, data, and network," 2012.

[8] A. Luckow, L. Lacinski, and S. Jha, "Saga bigjob: An extensible and interoperable pilot-job abstraction for distributed applications and systems," *Cluster, Cloud and Grid Computing (CCGrid)*, 2010.