Performance Summary of RADICAL-Pilot YARN

Nikhil Shenoy

May 10, 2016

Abstract

While RADICAL-Pilot and the EnsembleMD Toolkit are self-contained software packages, their extensibility allows developers to utilize the performance benefits for other applications as well. For example, the areas of bio-molecular dynamics and genomics require capabilities for handling compute-intensive and data-intensive tasks, but RADICAL-Pilot can provide only some of this functionality. These fields require a combination of the best techniques from high performance computing and from data processing platforms like Hadoop in order to achieve good results. For this reason, RADICAL-Pilot has been extended to include Hadoop and the associated YARN resource management system to take advantage of RADICAL-Pilot's high performance computing applications and Hadoop's data management capabilities. In this paper, we compare the performance of RADICAL-Pilot with RADICAL-Pilot extended with YARN to demonstrate the usefulness of the additional functionality.

1 Introduction

RADICAL-Pilot is a Python API developed by the RADICAL-Cybertools group that aids developers in submitting and running batches of tasks on high performance machines. The API achieves this through a container called a Pilot; this container is assigned the information associated with each task in the batch, such as the location of input data and what simulation to run, and is then placed in the queue of an HPC machine. Once the scheduler on the HPC machine schedules the Pilot onto a resource, the Pilot will then start its own Agent to begin carrying out the batch of tasks. It aggregates all the necessary resources, and then pulls additional information about each task from MongoDB. The Agent is responsible for carrying out the execution for each task and making sure that the output is sent back to the user.

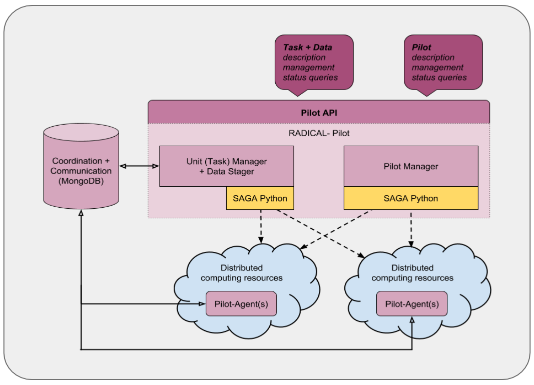

Figure 1: The RADICAL-Pilot Architecture [1]

This method of scheduling tasks provides many advantages, one of which is circumventing the scheduler. In current submission scenarios, each task must occupy a place in the scheduling queue and must individually wait for the required resources to become available before executing. This greatly increases the total time spent in the queue for the collection of tasks, which also increases the time to completion. However, Pilots allow users to avoid this time penalty by taking advantage of late binding and sending only the Pilot through the scheduler, reducing the wait time to that of a single task. Once the Pilot is finally scheduled, the user can then employ the various execution styles provided by the API based on the nature of their application. For example, if the batch is split into two types of tasks where one must occur before the other, the API provides functionality to schedule and execute such chained tasks. This, and the other options provided by RADICAL-Pilot, presents the user with a simple but powerful interface for task execution that outstrips current methods. By permitting any executable to be associated with a task in a Pilot, the RADICAL-Pilot API is flexible enough to handle a variety of HPC tasks, making it ideal for users who regularly work on the order of thousands of simulations.

While RADICAL-Pilot offers an improvement in efficiency for HPC tasks, it is not designed to handle data-intensive tasks. However, Apache Hadoop and YARN are able to work on problems involving large amounts of data. YARN in particular plays the roles of resource manager and scheduler in this context, and contributes the main functionality for wrestling with large volumes of data [2]. RADICAL-Pilot has been extended with YARN in order present an iterface which has performs well on HPC and data-intensive tasks. The extension has been implemented at the level of RADICAK-Pilot's Agent, the entity responsible for coordinating execution on the remote machine, within several components. The Local Resource Manager now retrieves new environment variables that detail the number of cores to be used on each node and the assignment of nodes, among other parameters. It then passes this information on to the newly started Hadoop and YARN demons, which examine and record the current state of the cluster. The RADICAL-Pilot scheduler has also been updated to include information about the current state of the cluster, including updates on the total memory available and the total number of cores in use. The scheduler then uses this state information to schedule the next task appropriately. Finally, the Application Manager, which handles the resource allocations, works with the YARN Resource Manager in order to coordinate the execution of tasks. RADICAL-Pilot provisions a Compute Unit with a Description that contains the resource requirements, and then requests that YARN create a container for it. By placing the Compute Unit within the YARN container, the YARN scheduler can then easily assign the container the optimal resources for execution [3].

YARN Image here

These extensions to RADICAL-Pilot give birth to they RADICAL-Pilot-YARN package, which contains functionality to operate on HPC and data-intensive tasks. In this paper, we will examine the performance of RADICAL-Pilot-YARN in comparison to RADICAL-Pilot by itself. We will do this through a simulation with a simple clustering algorithm.

2 Experiments

Our experiments to compare the two softwares involve running the K-means algorithm using a varying number of clusters and points in order to judge performance capabilities. We first summarize the algorithm, and then detail the experiment configuration in Table 1.

2.1 Algorithm and Configuration

The K-means algorithm seeks to iteratively classify a set of data points in to k different clusters. It starts by placing k centroids as far away from each other as possible, but still making a first approximation of the possible classifications. Then every data point is assigned to the nearest centroid based on distance. Once this is done, a new set of k centroids is determined based on the previous assignments [4]. Over many iterations, the centroids will eventually converge to their proper locations. The iteration is done by minimizing an error function:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \|\mathbf{x}_i^{(j)} - \mathbf{c}_j\|^2$$

where $\|\mathbf{x}_i^{(j)} - \mathbf{c}_j\|^2$ represents the distance between centroid $\mathbf{c}_j$ and the data point $\mathbf{x}_i^{(j)}$. The result are k different points that represent the optimal classifications fo the data set.

Parameter	Value
RADICAL-Pilot Version	v0.40.1-41-g6101f4a
Target Machine	XSEDE Stampede
Iterations of K-Means	2
Cores	[8,16,32]
Tasks	[8,16,32]
Data sets	`dataset_100K_3d.in`, `dataset_10K_3d.in`, `dataset_1M_3d.in`

Table 1: Experiment Parameters

Table 1 shows the parameters involved in this experiment. We implement the k-Means algorithm on XSEDE's Stampede machine using RADICAL-Pilot version 0.40.1, and we run two iterations of the algorithm. We want the computational cost of the experiment to remain the same in each configuration, so we constrain the product of the number of clusters and the number of data points to be constant. Doing this, we run the algorithm on configurations of 50 clusters and 1,000,000 points, 500 clusters and 100,000 points, and 5000 clusters and 10,000 points. Each of these points is represented by a vector of length three, which is commonly used to describe the locations and velocities of particles in three dimensional space. We run each configuration using 8, 16, and 32 cores each, and we set the number of tasks equal to the number of cores. Finally, we run this entire configuration using both RADICAL-Pilot and RADICAL-Pilot-YARN. This set up of the experiment allows us to examine the scaling behavior of both softwares in terms of the number of cores, and to compare the performance of each under identical conditions.

In this experiment, we only track the time-to-completion for each software. Measuring individual components is not necessary here, as the experiment is intended to present the two softwares as black boxes and requires the user to pay attention only to how fast his script is completed. However, one can see what states the time-to-completion measurement encompasses in Figure 2.

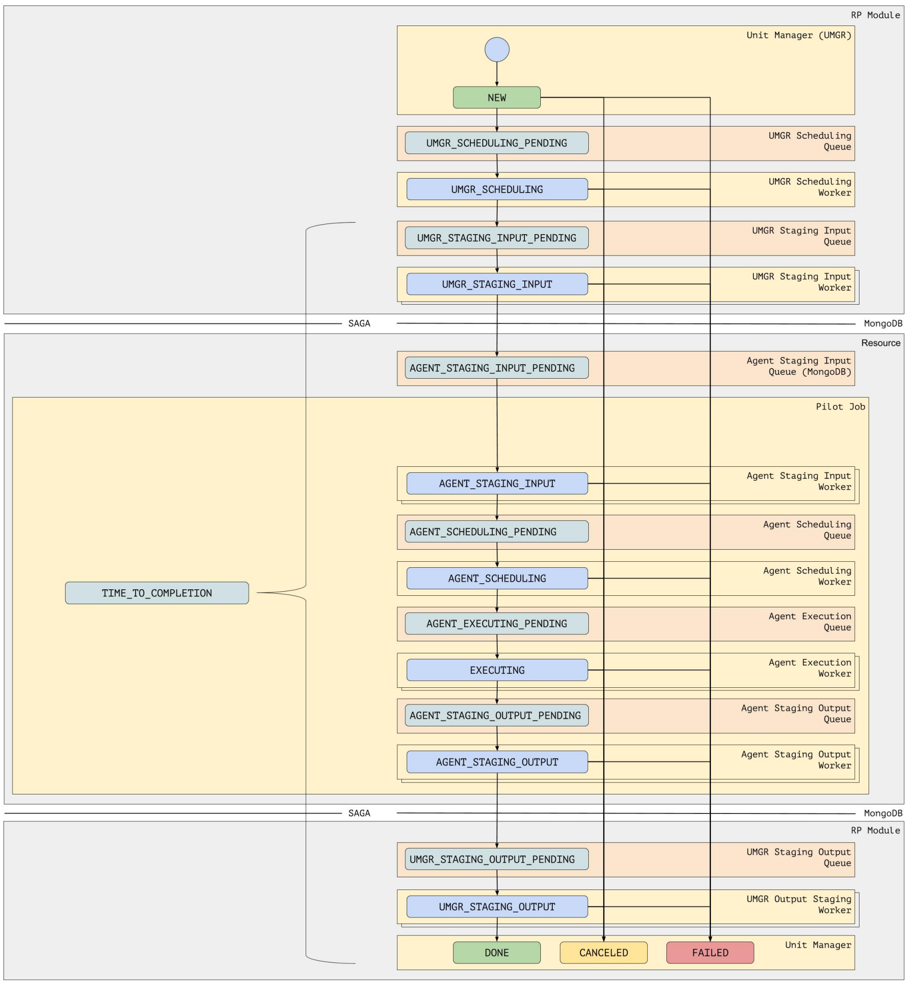

Figure 2: Mapping of parameters to pilot state model. Derived from [5].

As shown in Figure 2, the time-to-completion encompasses almost all of the states that the Pilot transitions through. This includes the staging of input for the Unit Manager and Agent, scheduling and executing simulations, and then staging the resultant data out to the Agent and Unit Manager. Formally, we state the time-to-completion in Equation 1:

$$Time-to-Completion = DONE - UMGR\_STAGING\_INPUT\_PENDING \tag{1}$$

This measurement, taken in seconds, provides us with the time necessary to complete a user's tasks.

2.2 Analysis

For each configuration of points and clusters, we graphed the time-to-completion across all cores.

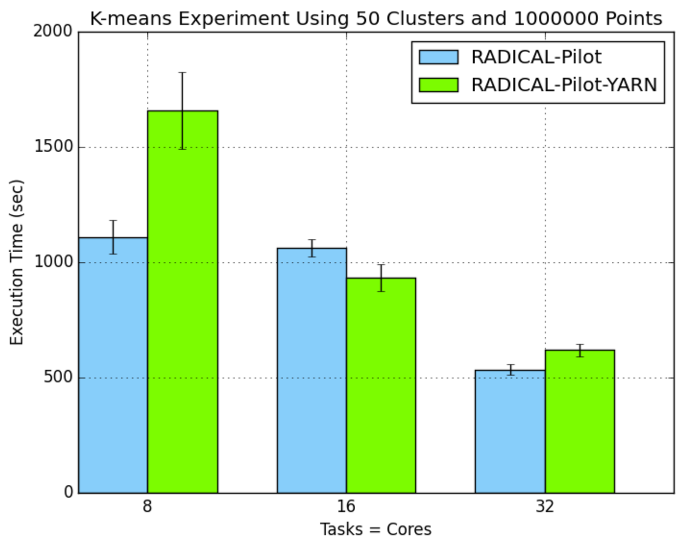

Figure 3: 50 clusters, 1,000,000 points

In the configuration with 50 clusters, the first trend that we noticed is that the time-to-completion decreases as the core count increases. This is expected, since holding the total amount of computation the same and increasing the core count classifies this experiment as an instance of strong scaling. We observe this for both softwares. However, the comparisons between RADICAL-Pilot and RADICAL-Pilot-YARN do not yield a consistent pattern of increased performance in favor of RADICAL-Pilot-YARN. In the case with 8 cores, the time-to-completion for RADICAL-Pilot-YARN far outstrips that of RADICAL-Pilot, which is counter to the design goals RP-YARN. Scaling up to 16 cores, we notice that the relationship between the two softwares is reversed. RP time-to-completion exceeds that of RP-YARN in a range of about 20-200 seconds, within error bars. The comparison is decidedly in favor of RP-YARN, since it's time-to-completion is lower. In the case with 32 cores, the relationship reverts back to being in favor of RP. Yet, contrast in the execution times is much less than it was in the first case, indicating a difference of at most 200 seconds.

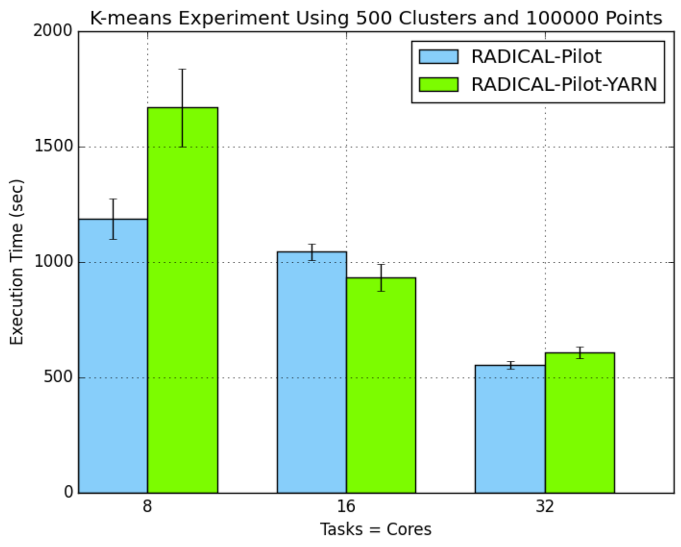

Figure 4: 500 clusters, 100,000 points

Figure 4 shows the scaling behavior for the 500 cluster, 100,000 point case. The plots for the 500 and 50 cluster cases are remarkably similar. The 8 core case showed an increase in execution time for both RP and RP-YARN in comparison to the 50 cluster, 8 core case, but otherwise showed a similar disparity of time between the two softwares. RP execution time decreased slightly between the configurations for the 16 core case, but error bars indicate that RP statistically still takes longer than RP-YARN to run. Finally, the 32 core case shows a decrease in execution time for RP but an increase for RP-YARN, accentuating the relationship observed in the 50 cluster configuration. In summary, while there were fluctations in the times-to-completion of each execution, this configuration of cluster and points reinforced the trends noticed in the 50 cluster case.

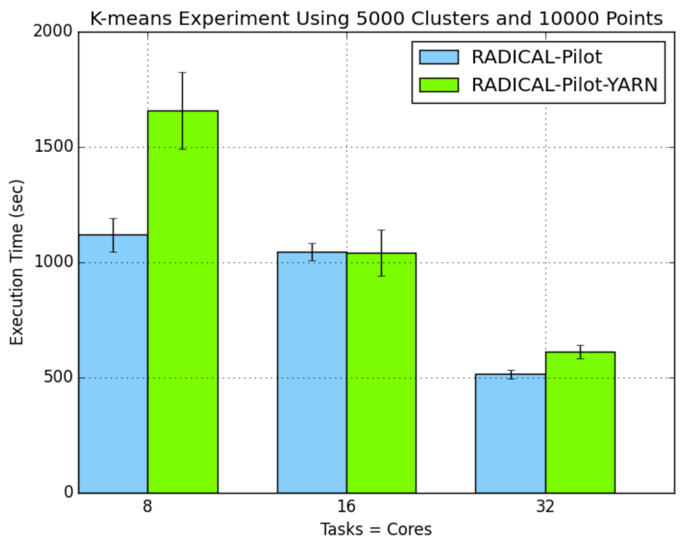

Figure 5: 5000 clusters, 10,000 points

The 5000 cluster case also showed similarities to the trends in the 50 cluster and 500 cluster cases, as shown in Figure 5. Again, we see a large disparity in the execution times of RP and RP-YARN at 8 cores, ranging from 300 seconds to 700 seconds. The ratio between the two times relatively constant when compared to the 50 and 500 cluster cases. At 16 cores, we notice that the times-to-completion are almost identical, but the error bars give us more insight into the behavior. Since the error bar for RP-YARN is larger than that of RP, it is possible that the time-to-completion for RP-YARN may fluctuate between being greater than, less than, and equal to the time-to-completion for RP. At 32 cores, we see the largest difference of the three configurations between RP-YARN and RP, with a range of at most 200 seconds. Again, this supports the trends found in the other two cases.

3 Conclusion

4 Future Work

5 Lessons Learned

References

[1] RADICAL-Cybertools, “Radical-pilot architecture,” 2016. [Online]. Available: https://radical-cybertools.github.io/

[2] A. S. Foundation, “Apache hadoop yarn,” 2016. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[3] A. Luckow, I. Paraskevakos, G. Chantzialexiou, and S. Jha, “Hadoop on hpc: Integrating hadoop and pilot-based dynamic resource management,” *arXiv*, 2016. [Online]. Available: http://arxiv.org/abs/1602.00345

[4] M. Matteucci, “A tutorial on clustering algorithms: K-means clustering,” 2016. [Online]. Available: http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html

[5] RADICAL-Cybertools, “Radical-pilot global state model,” 2016. [Online]. Available: https://radicalpilot.readthedocs.org/en/stable/_images/global-state-model-plain.png