Model-driven Autoscaling for Hadoop clusters

Anshul Gandhi
Stony Brook University
anshul@cs.stonybrook.edu

Parijat Dube, Andrzej Kochut, Li Zhang
IBM T. J. Watson Research Center
{pdube,akochut,zhangli}@us.ibm.com

I. Introduction

The growing need for **data processing and analytics (DPA)** has resulted in the development of a broad set of tools specifically tailored for data analysis, such as Hadoop and Spark. Cloud service providers, recognizing this emerging trend, have started incorporating DPA in their cloud offerings, for example, Elastic MapReduce from Amazon.

Cloud users are typically interested in minimizing their resource rental costs while completing their jobs in the necessary time frame (execution time SLA). Unfortunately, this is a very difficult task for several reasons: (1) The performance of DPA applications depends on many internal and external parameters (for example, Ganglia provides 200+ metrics for monitoring Hadoop), and these relationships are usually complex and workload-dependent; (2) The dynamic, shared nature of cloud computing often results in unpredictable application performance due to, for instance, node failures and resource contention; (3) DPA applications are typically composed of multiple stages, each of which requires different resource allocations. Efficient utilization of resources would require the ability to *dynamically scale* the deployment *while* the job is in progress, and the expertise to determine the required scaling for SLA compliance.

Prior research does not completely address the above challenges. While there has been recent work [1], [2] on performance modeling of DPA applications, such models are not well suited for dynamic scaling. Recent work [3], [4] on dynamic management of DPA applications focuses on redistributing existing resources without addressing dynamic scaling for SLA compliance. There has also been recent work on autoscaling in cloud environments [5], [6]; however, these works focus on transactional (non-DPA) workloads.

In this paper, we focus specifically on enabling autoscaling for DPA applications on Hadoop. We first derive simple yet powerful performance models for various DPA workloads in Section II. Our models only focus on important system parameters, providing very good accuracy (typically less than 5% error) without being overly complex. We then leverage these models in Section III to decide the scaling actions needed to meet the execution time SLAs under a range of scenarios, including node failures. Our implementation results on an OpenStack-deployed cloud environment demonstrate the efficacy of our autoscaling solution.

II. Modeling

We employ a model-driven approach to autoscaling. Our modeling methodology involves profiling a Hadoop workload under various scenarios, such as varying job and cluster configurations, and then using regression to build simple, descriptive models of execution time as a function of, among others, resource allocation.

Based on the characteristics of Hadoop and its job execution, we model job execution time, T_{job} (in secs), as a function of input data size, D (in MB), number of Map and Reduce tasks, M and R, number of Map and Reduce configured cores, N_{mc} and N_{rc}, and the number of Map and Reduce slots per core, n_{ms} and n_{rs}. We then use training data to fit our generic models to specific workloads.

Workloads: We employ three different Hadoop workloads for our experiments: WordCount (dominated by the Map stage, CPU-bound), TeraSort (CPU-bound Map stage and an I/O-bound Reduce stage), and Kmeans (multi-job workload, composed of multiple CPU-bound Iteration jobs and a single I/O-bound Classification job).

Performance Models: We separately model the Map stage execution time, T_{ms}, and the Reduce stage execution time, T_{rs}. Note that $T_{job} = T_{ms} + T_{rs}$. Due to lack of space, we omit the model derivations and present the final models.

WordCount:

$$T_{ms} = \left(0.4\frac{D}{M} + 6\right) \cdot \left\lceil \frac{M}{N_{mc} \cdot n_{ms}} \right\rceil \cdot \min\left(\left\lceil \frac{M}{N_{mc}} \right\rceil, n_{ms} \right)$$

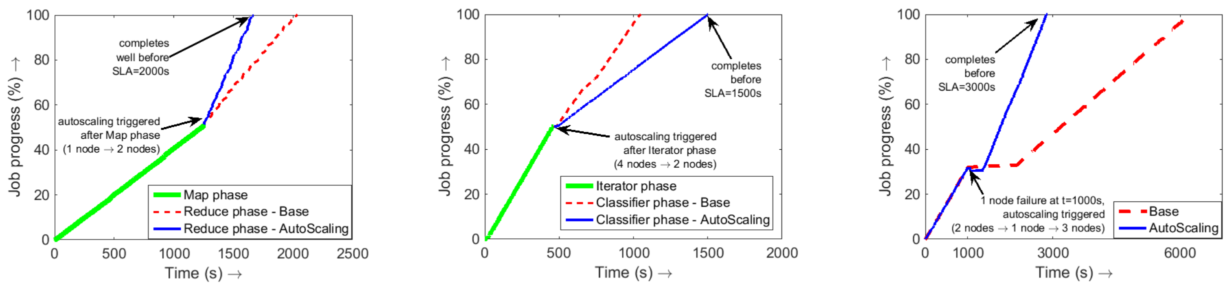
$$T_{rs} = \left[5 + (0.5E-3)\frac{D}{R} + \left(6 + (7E-3)\frac{D}{R}\right) \cdot \left(\left\lceil \frac{R}{N_{rc} \cdot n_{rs}} \right\rceil - 1 \right) \right] \cdot \min\left(\left\lceil \frac{R}{N_{rc}} \right\rceil, n_{rs} \right) + 0.1\frac{M}{R}$$

TeraSort:

$$T_{ms} = \left(0.6\frac{D}{M} - 29\right) \cdot \left\lceil \frac{M}{N_{mc} \cdot n_{ms}} \right\rceil \cdot \min\left(\left\lceil \frac{M}{N_{mc}} \right\rceil, n_{ms} \right)$$

$$T_{rs} = \left[\left((62E-3)\frac{D}{R} + (5E-6)\left(\frac{D}{R}\right)^2 \right) \cdot \left(\left\lceil \frac{R}{N_{rc} \cdot n_{rs}} \right\rceil - 1 \right) + 16 + (39E-3)\frac{D}{R} \right] \cdot \min\left(\left\lceil \frac{R}{N_{rc}} \right\rceil, n_{rs} \right) + 3.5\frac{M}{R}$$

The `min` term above accounts for the corner cases where M is less than N_{mc}. The M/R term accounts for the overhead of data movement in the Shuffle phase.



(a) Scaling up capacity for TeraSort. (b) Scaling down capacity for Kmeans. (c) Scaling for failure recovery for WordCount.

Figure 1. Implementation results illustrating our autoscaling approach for various use cases and workloads. In all cases, we successfully meet the SLA.

The execution time for Kmeans depends on the number of Kmeans clusters (K_c), which is an input parameter. Since the above models do not account for such parameters, we directly model execution time as a function of input data size, D, number of Kmeans clusters (K_c), and number of Map cores, N_{mc}. We model job execution time of the two Kmeans phases (Iteration and Classification) separately. In our experiments, we find that the execution time of the (non-overlapping portion of) Reduce stage in the two phases of Kmeans is negligible (in fact, the Classification phase is Map-only). We thus ignore the Reduce stage execution time.

$$\begin{array}{ll}\text{Iteration} & T_{job} = \left(2.1K_c + 0.04K_c^2 + 0.1K_cD\right)/N_{mc} \\ \text{Classification} & T_{job} = \left(2.4K_c + 0.05K_c^2 + 0.2K_cD\right)/N_{mc}\end{array}$$

Observe that Kmeans execution time has a quadratic dependence on K_c, and also depends on $K_c \times D$.

Validation: The regression models obtained above exhibit a good fit, with R^2 (goodness of fit coefficient) values above 0.97. For validation, we employ the above models to estimate T_{job} for test data, which is different (in terms of job and/or cluster configuration) from training data. Our modeling error over test data is less than 10%, with an average error of about 4%.

III. Evaluation

We now leverage the above workload-specific models for autoscaling resources needed by a given job to meet its execution time SLA. For each experiment, we first illustrate performance using the default approach, **Base**, that does not scale capacity. We then repeat the experiment and, to highlight our solution, dynamically invoke our approach, **AutoScaling**, *while* the job is in progress.

Experimental Setup: We set up Hadoop (version 1.2.1) with 1 Master node VM and multiple 1-core 4GB memory Slave node VMs (with 1 slot/node) on an OpenStack managed private cloud deployment with multiple 8-core 64GB memory hypervisors. To add a new Slave node, we boot a new VM using a pre-loaded Hadoop image, and then start the TaskTracker and DataNode services on it. To remove a Slave node, we simply update the exclude list on the Master. The Master then migrates the HDFS data, if any, from this node. After this migration, the node can be turned off.

Experimental Results: Fig. 1 shows our experimental results for all workloads under different scenarios. We monitor job progress via Hadoop logs. Since autoscaling is invoked while the job is in progress, we set D as the remaining amount of data to be processed (estimated via Hadoop progress logs). Fig. 1(a) illustrates the scenario where we scale up capacity just after execution of the Map stage to ensure that the SLA is met for TeraSort. Fig. 1(b) illustrates the scenario where we scale down capacity between different job phases of Kmeans to save on rental costs (or resource usage) while ensuring SLA compliance. Lastly, Fig. 1(c) illustrates the interesting case of performance recovery following a node failure for WordCount. The loss in performance is detected via progress logs, which results in invocation of our autoscaling solution that adds the required capacity to meet the SLA. The stall in progress (flat lines around the 1000s mark) is due to re-execution of lost tasks resulting from node failures. Note that, in all cases, Base does *not* meet the SLA whereas AutoScaling does.

IV. Conclusion

In this paper, we present our solution for agile autoscaling of cloud-deployed DPA clusters that relies on our novel, workload-specific performance models. We hope that our solution promotes efficient cloud analytics, enabling the availability of systems-of-insights to a broader audience.

References

[1] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments," in *Proceedings of ICAC 2011*.

[2] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A Self-tuning System for Big Data Analytics," in *Proceedings of CIDR 2011*.

[3] W. Zhang, S. Rajasekaran, T. Wood, and M. Zhu, "MIMP: Deadline and Interference Aware Scheduling of Hadoop Virtual Machines," in *Proceedings of CCGrid 2014*.

[4] B. Ghit, N. Yigitbasi, A. Iosup, and D. Epema, "Balanced Resource Allocations Across Multiple Dynamic MapReduce Clusters," in *Proceedings of SIGMETRICS 2014*.

[5] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, Model-driven Autoscaling for Cloud Applications," in *Proceedings of ICAC 2014*.

[6] ——, "Modeling the Impact of Workload on Cloud Resource Scaling," in *Proceedings of SBAC-PAD 2014*.