Towards Multi-Objective Optimisation of Hadoop 2.x Application Deployment on Public Clouds

Naif Alasmari
Department of Computer Science
University of York
York, United Kingdom
nnma500@york.ac.uk

Radu Calinescu
Department of Computer Science
University of York
York, United Kingdom
radu.calinescu@york.ac.uk

***Abstract*—Hadoop is a widely-used software platform for the development, deployment and execution of Big Data applications. Leading technology companies such as Yahoo and Facebook are regularly employing Hadoop to process large datasets. Nevertheless, running Hadoop applications with effective performance-cost trade-offs is very challenging due to the large number of Hadoop parameters that need to be appropriately configured. The challenge is compounded by the frequent practice of deploying Hadoop applications on public cloud infrastructure, as this also requires the selection of suitable cloud configuration parameters (e.g., types and number of virtual machines) for each application. To address this challenge, our work-in-progress paper proposes an approach for the multi-objective optimisation of the Hadoop and cloud parameters of Hadoop 2.x applications deployed on public clouds. Our approach uses Hadoop and cloud infrastructure models to synthesise sets of configurations that achieve Pareto-optimal trade-offs between the execution time and the cost of Big Data applications, enabling users to select optimal deployments that meet their time and/or budget constraints.**

***Index Terms*—Big Data, Hadoop, MapReduce, Cloud computing, Multi-objective Optimisation.**

I. Introduction

The fast development of cloud, mobile, IoT and other technologies has led to a significant rise in the generation, exchange, storing and processing of data [1]–[3]. Around 2.5 exabytes (10^{18} bytes) of data were produced daily in 2012, and this quantity is almost doubling every forty months [4]. This increase in the generation of data in many important applications has greatly surpassed what traditional data processing tools can handle in acceptable time [5]. Introduced by Google in 2004, MapReduce [6] is a widely-used programming model that addresses this problem by supporting the processing of very large datasets in two main steps [7]. In the former step, a *map* function is applied by multiple *worker nodes* to subsets of the input data organised into (key, value) pairs. In the latter step, the intermediate results produced by the map, also formatted as (key, value) pairs, are *shuffled* so that pairs with the same key end up on the same worker node, and are combined using a *reduce* function that produces the final results. Hadoop is an open-source MapReduce implementation broadly adopted within academia and industry [8], [9]. This implementation broke a world record in May 2009 by sorting one terabyte in 62 seconds and one petabyte in 16.25 hours [1]. However, the latest stable versions of Hadoop (i.e., Hadoop 2.x) have over 200 interdependent configuration parameters [10], most of which influence the performance of Hadoop applications. Manually initialising these parameters is a tedious, error-prone process that is further complicated by the common practice of deploying Hadoop applications on cost-effective public cloud infrastructure, where the types and number of virtual machines to use also need to be decided. As such, configuring cloud-deployed Hadoop 2.x applications to meet the time and/or budget constraints of their users is very challenging.

To address this challenge, we propose an approach for the automated synthesis of optimal configurations for cloud-deployed Hadoop 2.x applications. Our Hadoop Configuration Optimizer (HCO) approach uses:

1) a performance model of the Hadoop 2.x application being deployed and a cloud cost model to accurately predict the execution time and cost of different configurations;
2) multi-objective optimisation techniques to efficiently identify configurations that achieve Pareto-optimal trade-offs between execution time and cost.

This work-in-progress paper provides an overview of the new approach, and summarises our preliminary results and next steps of our project.

The rest of the paper is organised as follows. Section II describes existing research for predicting the cost and execution time of Hadoop applications, and compares them to our approach. Next, Section III presents the steps involved in using our HCO approach, and the internal architecture of our solution. Sections IV and V conclude the paper with a summary of the results we obtained so far and with a brief discussion of the next steps of our project, respectively.

II. Related Work

Several methods have been proposed for predicting or optimising the cost and/or execution time of Hadoop applications. Chen et al. [11] devised a Hadoop 1.x mathematical model supporting the cloud execution of a Hadoop job within a time bound and with the lowest possible cost. Verma et al. [12] proposed a management decision mechanism enabling Hadoop 1.x workloads to meet the deadlines constraints of Hadoop jobs. Furthermore, Zacheilas et al. [13] introduced a

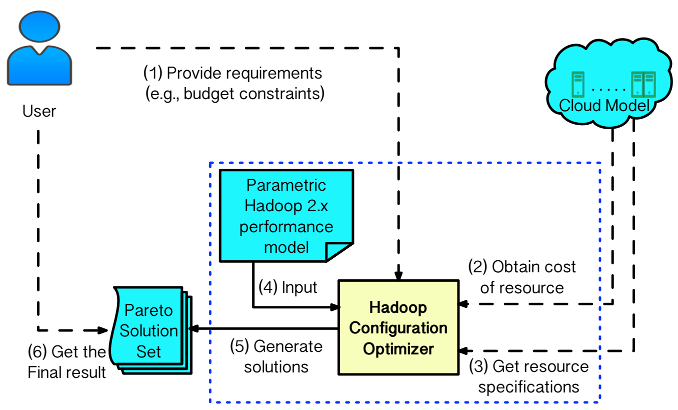
User
(1) Provide requirements
(e.g., budget constraints)
Cloud Model
Parametric Hadoop 2.x performance model
(4) Input
Hadoop Configuration Optimizer
(2) Obtain cost of resource
(3) Get resource specifications
(5) Generate solutions
Pareto Solution Set
(6) Get the Final result

Fig. 1. Hadoop Configuration Optimizer workflow

Hadoop 1.x scheduler that can achieve Pareto-optimal trade-offs between cost and performance. However, all these methods are based on Hadoop 1.x performance models, which assume static allocation of resources within the MapReduce steps, whereas Hadoop 2.x resource allocation is dynamic. Our new approach aims to capture this significant change in resource allocation between Hadoop 1.x and Hadoop 2.x, and thus to support the configuration of Big Data applications that use the latest stable versions of Hadoop.

Other recent solutions do not consider important factors that influence the performance of cloud-deployed Hadoop applications. The steady-state non-dominated sorting genetic algorithm used by [14] tackles the optimisation problem without taking into account the cost of I/O operations, which significantly affect on the cost of cloud-deployed Hadoop applications. Similarly, the performance model introduced by Lin et al. [15] to predict the performance of MapReduce tasks does not consider the competition for resources between concurrent map and reduce tasks. Finally, Song et al.'s method for predicting the performance of Hadoop jobs by using locally weighted regression methods [10] does not consider the cost of executing Hadoop applications. Our approach aims to tackle these limitations, by considering all major factors that influence the performance and cost of Hadoop applications.

III. Hadoop Configuration Optimizer Approach

A. User perspective

Fig. 1 shows how a user of the HCO approach can take advantage of its automated configuration synthesis. In Step 1, the user provides essential HCO inputs including: the Hadoop application to be deployed; the location of the dataset to be processed; a selected public cloud provider; and any budget and/or execution-time constraints the user may have.

Next, HCO obtains the cost of different types of virtual machines (VMs) available from the selected cloud provider and technical specifications for these VM (e.g., number of CPU cores and the size of RAM) in Steps 2 and 3, respectively. This information is extracted from a predefined cloud cost model, available from a HCO repository of such models for known cloud providers.

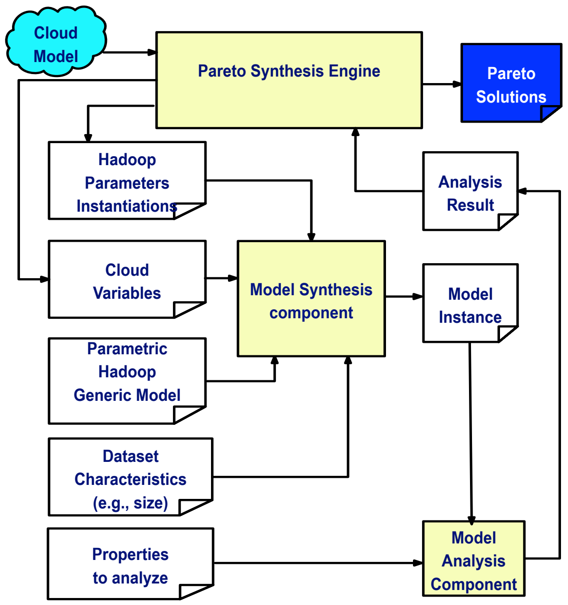

Fig. 2. HCO architecture

In Step 4, our HCO approach builds a performance model for the Hadoop application under deployment. This model is obtained by using the user-provided information from Step 1 and the cloud resource cost and specification information from Steps 2 and 3 to initialise the relevant parameters of a generic HCO parametric performance model of cloud-deployed Hadoop 2.x applications. The result is an application-specific, *partial instantiation* of the generic parametric Hadoop 2.x performance model, and still contains two types of yet-to-be-determined parameters: the Hadoop configuration parameters and the cloud resource parameters for the deployment of the application.

Pareto-optimal combinations of values for the two types of parameters mentioned above are then synthesised in Step 5 of our HCO approach. Finally, the resulting set of Pareto-optimal configurations is presented to the users in Step 6, so they can select a configuration that not only meets their initial time and/or budget constrains, but also achieves the most attractive performance-cost trade-off.

B. Hadoop Configuration Optimizer Architecture

Fig. 2 depicts the internal architecture of our Hadoop Configuration Optimizer, with its three core components, each of which is briefly described below.

1) The Model Synthesis Component combines

- inputs received from the user (i.e., application and dataset characteristics)
- inputs from the HCO Pareto-front synthesis engine (i.e., specific Hadoop 2.x and cloud infrastructure configurations for which execution time and cost predictions are needed)

to produce a fully instantiated performance model of the Hadoop application to be deployed. Our HCO approach is not prescriptive about the modelling paradigm used to formally model the behaviour of a Hadoop application, and can equally support the use of stochastic models (e.g., continuous-time Markov chains or queueing networks) or (as illustrated in Section IV) analytical models.

2) The Model Analysis Component predicts the execution time and cost of the Hadoop application configuration specified by the model instance sythesised by the previous HCO component. The analysis result is sent to the Pareto Synthesis Engine.

3) The Pareto Synthesis Engine generates a set of Pareto-optimal configurations that achieve optimal trade-offs between the performance and cost of cloud-deployed applications. To explore the very large configuration space of a Hadoop application, the component uses multi-objective optimisation metaheuristics (e.g., evolutionary algorithms) as in our recent work on the synthesis of stochastic models [16]–[19]. These are optimisation methods where a *population* of solutions is improved over a number of iterations (called *generations*) by retaining and combining the best solutions from each step, i.e., the solutions that satisfy the user constrains, and are either Pareto-optimal or belong to an under-represented area of the configuration space. The search process is executed until a termination criterion is met. Typical termination criteria for metaheuristic optimisation include stopping after a predetermined number of iterations, or stopping after several successive iterations that provide only negligible improvements.

IV. Preliminary Model and Experimental Results

A. Initial Analytical Model

We have so far devised a simple parametric Hadoop 2.x performance model (cf. Fig. 1) to assess the effectiveness of our HCO approach. This model uses the 15 key Hadoop 2.x parameters shown in Table I, first to compute a set of derived Hadoop application parameters (also listed in Table I), and then to predict the execution time and cost of the application under deployment.

Our model supports both methods used by Hadoop to compute the number of containers for a Big Data application. The first method, called *deFault Resource Calculator* (FRC), calculates the number of map containers (M_{cont}) and the number of reduce containers (R_{cont}) based solely on the total size of memory for the cluster; the number of CPU cores is not used in the calculation. The second method, called *Dominant Resource Calculator* (DRC), takes into account both the memory size and the number of CPU cores. Thus, given a type of calculation method $Cal_{type} \in \{FRC, DRC\}$, these numbers of containers are defined by:

$$\begin{cases} M_{cont} = C_{capacity} \left\lfloor \frac{NM_m}{\max(M_m, Ymin_m)} \right\rfloor \\ R_{cont} = C_{capacity} \left\lfloor \frac{NM_m}{\max(R_m, Ymin_m)} \right\rfloor \end{cases} \quad (1)$$

if $Cal_{type} = FRC$, and by

$$\begin{cases} M_{cont} = C_{capacity} \min \left\{ \left\lfloor \frac{NM_m}{\max(M_m, Ymin_m)} \right\rfloor, \left\lfloor \frac{NM_c}{\max(M_c, Ymin_c)} \right\rfloor \right\} \\ R_{cont} = C_{capacity} \min \left\{ \left\lfloor \frac{NM_m}{\max(R_m, Ymin_m)} \right\rfloor, \left\lfloor \frac{NM_c}{\max(R_c, Ymin_c)} \right\rfloor \right\} \end{cases} \quad (2)$$

TABLE I
Parameters of the initial Hadoop 2.x performance model

Parameter	Notation
Hadoop configuration parameters	
yarn.scheduler.capacity.resource-calculator	Cal_{type}
yarn.scheduler.minimum-allocation-mb	$Ymin_m$
yarn.scheduler.minimum-allocation-vcores	$Ymin_c$
yarn.scheduler.maximum-allocation-mb	$Ymax_m$
yarn.scheduler.maximum-allocation-vcores	$Ymax_c$
yarn.nodemanager.resource.memory-mb	NM_m
yarn.nodemanager.resource.cpu-vcores	NM_c
mapreduce.map.memory.mb	M_m
mapreduce.map.cpu.vcores	M_c
mapreduce.reduce.memory.mb	R_m
mapreduce.map.cpu.vcores	R_c
dfs.blocksize	B_{size}
mapred.max.split.size	$Split_{max}$
mapred.min.split.size	$Split_{min}$
mapreduce.job.reduces	R_{job}
Application parameter	
Dataset size	D
Cloud parameters	
Cluster Capacity (number of nodes)	$C_{capacity}$
Cost of node per hour	N_{cost}
Derived parameters	
Allocated containers for Map	M_{cont}
Allocated containers for Reduce	R_{cont}
Number of Map tasks	M_{task}
Number of Reduce tasks	R_{task}
Rounds for Map	$Round_{Map}$
Rounds for Reduce	$Round_{Reduce}$
Execution time	E_{time}
Average Execution time for map	M_{time}
Average Execution time for reduce	R_{time}
Cost of execution time	cost

otherwise (i.e., if $Cal_{type} = DRC$), where the parameters from (1) and (2) are defined in Table I.

Once the number of containers is determined, the number of map tasks is computed as

$$M_{task} = \left\lceil \frac{D}{max(Split_{min}, min(Split_{max}, B_{size}))} \right\rceil, \quad (3)$$

B_{size}, $Split_{max}$, and $Split_{min}$ are configurable Hadoop parameters, and D is the size of dataset to be processed by the Hadoop application. Next, the numbers of rounds of map and reduce tasks are determined as

$$Round_{Map} = \left\lceil \frac{M_{task}}{M_{cont}} \right\rceil, \quad Round_{Reduce} = \left\lceil \frac{R_{task}}{R_{cont}} \right\rceil. \quad (4)$$

Finally, we estimate the application execution time E_{time} based on the average times of map and reduce phases:

$$E_{time} = (M_{time} * Round_{Map}) + (R_{time} * Round_{Reduce}) + \alpha, \quad (5)$$

where α represents the expected time consumed for setting up and releasing the containers, and for communication.

The cost paid for using the resources of cloud computing can then be calculated as:

$$cost = E_{time} * N_{cost} * C_{capacity}. \quad (6)$$

TABLE II
HCO PREDICTIONS VS. ACTUAL EXECUTION TIME

Setting Id	Real time	Estimated time	Error percentage
2	365.518	414.640	13.44%
3	366.503	361.829	1.28%
6	193.721	192.889	0.43%
8	198.812	171.240	13.87%
12	154.256	137.100	11.12%
13	227.485	265.600	16.75%

B. Preliminary Results

We carried out a set of experiments using six VMs with 4GB of RAM and 2 CPU cores each (i.e., one Name Node and five Data Nodes). To this end, we initially ran the Hadoop benchmark application WordCount with a 3GB dataset in one DataNode to get the times for map and reduce tasks, and the calibration factor α. The Hadoop configuration parameters were: Cal_{type} = FRC, NM$_m$ = 3072MB, B$_{size}$ and Split$_{max}$ = 128MB, M$_m$, R$_m$ and Ymin$_m$ = 1028MB, the number of reduce job is one. We obtained the cost of the single node per hour based on AWS calculator[1]. The cost was \$0.0464 (For the simplicity, we multiply it by 100). Then, we supplied the model with the following combination of parameter values: Split$_{max}$ = 64MB,128MB, 256MB, M$_m$ = 1024MB, 2048MB, Ymin$_m$ = 1024MB (in addition to the map and reduce task times that we obtained experimentally).

Running our prototype HCO implementation using these parameters generated the execution time and cost predictions from Fig. 3 for 27 possible Hadoop configurations. Four of these predictions (shown by filled circles) are Pareto optimal.

To assess the accuracy of the HCO predictions, we randomly selected six Hadoop configurations from Fig. 3, and ran real experiments using these configuration. The actual execution times from these experiments, compared to the HCO predictions in Table II, show that HCO achieved an acceptable accuracy even using the simple analytical model from our preliminary implementation.

V. Conclusion and Future Work

MapReduce and its implementation Hadoop support the execution of Big Data applications on cloud computing infrastructure. However, achieving acceptable trade-offs between the cost and execution time of these applications is challenging. In this paper, we presented an approach that uses a generic performance model of Hadoop 2.x to predict the execution time and cost of cloud-deployed Big Data applications. Based on these predictions, our approach generates an approximate Pareto-optimal set of Hadoop configurations.

The paper described our progress so far, and presented promising preliminary results. In our future work, we will experiment with larger volumes of data and multiple Big Data applications. Furthermore, we plan to develop a stochastic performance model of Hadoop 2.x and to compare its predictive power to that of the simple analytical model.

[1]https://calculator.s3.amazonaws.com/index.html

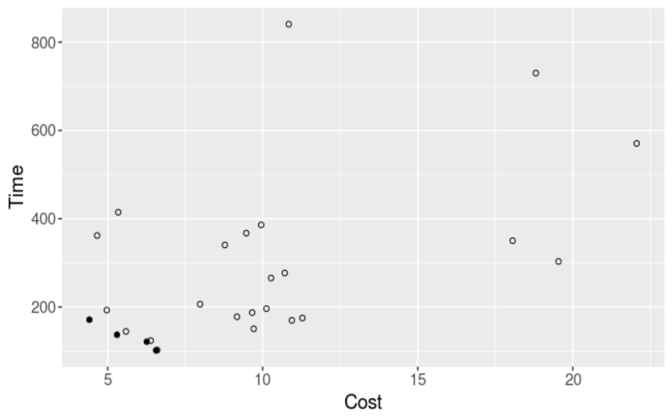

Fig. 3. Predicted execution time and cost for Hadoop configurations

References

[1] A. O'Driscoll, J. Daugelaite, and R. D. Sleator, "'big data', hadoop and cloud computing in genomics," *Journal of biomedical informatics*, vol. 46, no. 5, pp. 774–781, 2013.

[2] X. Zheng, P. Martin, K. Brohman, and L. D. Xu, "Cloud service negotiation in internet of things environment: A mixed approach," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 1506–1515, May 2014.

[3] C. Wang, Z. Bi, and L. Da Xu, "IoT and cloud computing in automation of assembly modeling systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1426–1434, 2014.

[4] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton, "Big data: the management revolution," *Harvard Business Review*, vol. 90, no. 10, pp. 60–68, 2012.

[5] L. D. Xu and L. Duan, "Big data for cyber physical systems in industry 4.0: a survey," *Enterprise Information Systems*, pp. 1–22, 2018.

[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Comm. of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] O. Hegazy, S. Safwat, and M. El Bakry, "A mapreduce fuzzy techniques of big data classification," in *SAI'16*, pp. 118–128, IEEE, 2016.

[8] N. Verma and J. Singh, "A comprehensive review from sequential association computing to hadoop-mapreduce parallel computing in a retail scenario," *Journal of Management Analytics*, vol. 4, no. 4, pp. 359–392, 2017.

[9] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining," in *ICDM'08*, pp. 512–521, IEEE, 2008.

[10] G. Song, Z. Meng, F. Huet, F. Magoules, L. Yu, and X. Lin, "A Hadoop Map-Reduce performance prediction method," in *HPCC_EUC'13*, pp. 820–825, IEEE, 2013.

[11] K. Chen, J. Powers, S. Guo, and F. Tian, "CRESP: Towards optimal resource provisioning for mapreduce computing in public clouds," *IEEE Trans. Parallel & Distrib. Systems*, vol. 25, no. 6, pp. 1403–1412, 2014.

[12] A. Verma, L. Cherkasova, V. S. Kumar, and R. H. Campbell, "Deadline-based workload management for mapreduce environments: Pieces of the performance puzzle," in *NOMS'12*, pp. 900–905, IEEE, 2012.

[13] N. Zacheilas and V. Kalogeraki, "Pareto-based scheduling of mapreduce workloads," in *ISORC'16*, pp. 174–181, IEEE, 2016.

[14] S. Imai, S. Patterson, and C. A. Varela, "Cost-efficient high-performance internet-scale data analytics over multi-cloud environments," in *CCGrid'15*, pp. 793–796, 2015.

[15] X. Lin, Z. Meng, C. Xu, and M. Wang, "A practical performance model for hadoop mapreduce," in *CLUSTER'12*, pp. 231–239, IEEE, 2012.

[16] R. Calinescu, M. Ceska, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Efficient synthesis of robust models for stochastic systems," *Journal of Systems and Software*, vol. 143, pp. 140–158, 2018.

[17] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering Journal*, May 2018.

[18] R. Calinescu, M. Ceska, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Designing robust software systems through parametric Markov chain synthesis," in *ICSA'17*, pp. 131–140, 2017.

[19] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in *ASE'15*, pp. 319–330, 2015.