Hadoop Branching: Architectural Impacts on Energy and Performance

Ivanilton Polato
Department of Computer Science
Federal University of Technology – Paraná
Campo Mourão, PR, Brazil
ipolato@utfpr.edu.br

Denilson Barbosa and Abram Hindle
Department of Computer Science
University of Alberta
Edmonton, AB, Canada
{denilson, abram.hindle}@ualberta.ca

Fabio Kon
Department of Computer Science
University of São Paulo
São Paulo, SP, Brazil
fabio.kon@ime.usp.br

***Abstract*—Data centers are notorious energy consumers. In fact, studies have shown that for every $1 spent on hardware in the datacenter, $0.50 is spent on powering this hardware over its lifetime. Data centers host real or virtual (i.e., cloud) clusters that often execute large compute jobs using MapReduce, of which Hadoop is a popular implementation. Like other successful open source projects, Hadoop has been maintained and evolved over time with new resource management features being added over time in an effort to improve performance, raising questions as to whether such architectural evolution has achieved its goal, and if so, at what cost. In this work we apply Green Mining to find out that later versions of Hadoop — who exhibit more dynamic resource control — can suffer from serious energy consumption performance regressions.**

I. Introduction

Apache Hadoop is an open-source, Java-based framework for distributed storage and computing that has quickly gained popularity for "big data" analytics applications in its 10-year history. It consists of a distributed file system (HDFS) and an implementation of Google's Map-Reduce [2] programming paradigm. Although hard statistics are hard to obtain, the number of commercial users of Hadoop is certainly in the thousands: mid-to-large size deployments typically have a few thousand compute nodes and storage capacity measured in petabytes[1]. Alone, IDC estimates that the Hadoop software market will be worth $813 million by 2016[2].

As most other highly successful open-source projects, Hadoop has been actively maintained and improved, and already has a quite extensive genealogy tree (Figure 1). Over time, a large number of features have been introduced and deprecated, resulting in three active development branches with several releases.

In this paper, we study the impact of such feature evolution w.r.t. resource usage and adoption by practitioners. More precisely, we deploy several representative and judiciously chosen versions of Hadoop on the same cluster, run a representative workload, and measure performance (execution time) and resource usage (power). Our results help clarify the cost-benefit trade-offs of the various versions of Hadoop.

Moreover, we mine the StackOverflow question/answer website to gather which versions of Hadoop are most frequently discussed. Using this measure as a proxy for feature adoption, and contrasting that with the corresponding cost-benefit trade-off, we arrive at some surprising observations. Most notably, our resource usage data shows that the recently "promoted"[3] YARN resource manager leads to *higher* energy consumption and *worse* performance, while we find that that users rarely mention or discuss it.

In parallel and distributed computing, resource usage is the key to achieve a well-balanced trade-off between performance increases and energy savings. Furthermore, resource usage can be closely related to the software architecture, since the latter determines how computation will be accomplished. Thus, changes in the software architecture during development — adding or removing features and components, modifications in classes, methods, and attributes — directly affects energy consumption.

Hadoop fits exactly this context. Over the last decade, Hadoop had more than 60 releases. Hadoop 1.x releases remained strictly correlated to MapReduce use, while 0.23.x and 2.x branches introduced critical architectural modifications, adding new components and changing the project architecture. Although these Hadoop releases became more flexible, the influence of such modifications on Hadoop's energy consumption was never fully analyzed in research.

Currently, these changes to the project's architecture have been overlooked by service providers — including cloud, storage, and elastic infrastructures. Therefore, the real impact of architectural changes on performance and energy consumption in frameworks such as Hadoop is unknown or ignored without further investigation. As the results of our investigation on Hadoop, our contributions include an energy consumption analysis of the Hadoop framework. We present the impact of architectural modifications in performance losses and on energy consumption increases.

II. Prior Work

Concerning Hadoop, a large portion of the recent studies is dedicated to modify and adapt Hadoop to solve a specific class of problems or to achieve better performance, generally by lowering the job makespan. Most of these adaptations are tied to specific Hadoop releases and require significant effort to work on later Hadoop releases. Most publications from

[1] http://www.enterprisetech.com/2013/11/08/cluster-sizes-reveal-hadoop-maturity-curve/

[2] http://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/

[3] http://adtmag.com/blogs/watersworks/2012/08/apache-yarn-promotion.aspx

978-1-5090-0172-9/15/$31.00

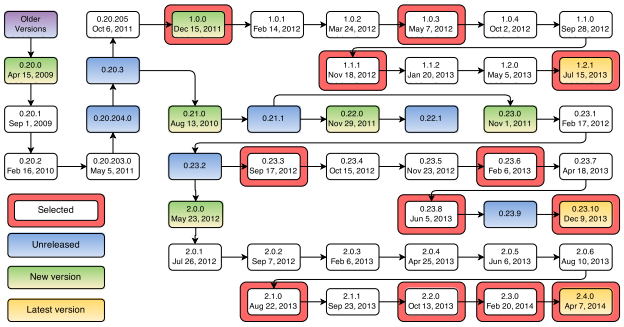

Figure 1. Hadoop Versions Genealogy Tree

2008 to 2013 covering the contributions to the framework were analyzed and described in a systematic literature review [8].

Although there is a constant concern with performance improvement, research on the energy impact of these modifications, especially in parallel and distributed platforms are rare. An example of a multi-version study of performance analysis was conducted on Hadoop by Shang et al. [9]. They studied Hadoop execution logs and their evolution over time.

Our work is motivated by the green mining methodology [5], [4], the study of energy consumption over multiple versions of a software system, in the area of energy-aware mining of Mining Software Repositories research [3]. The study presented here is a form of green mining applied to various versions of the Hadoop framework.

Multi-version energy consumption analysis has been applied by Hindle et al. [5] on numerous products. Others have tried to help energy driven UI refactorings: Li et al. [7] discuss optimization techniques to optimize UI colours for energy consumption.

III. Evidence of Changes in Hadoop's Performance

Hadoop evolved rapidly between 2011 and 2012. Architectural changes were introduced, bringing new components and consolidating flexibility into the framework. Behind the novelty from 0.23.x and 2.x branch releases hides a controversial dilemma in software engineering: performance versus flexibility. Is the loss in Hadoop's performance related to the architectural change necessary to implement YARN? Is it clear that the YARN resource manager has a direct negative impact on the Hadoop's performance?

Even though this issue can be confirmed by deploying releases from different branches, these questions remain virtually unanswered since the Hadoop community may have overlooked the impact of these modifications. The number of downloads and contributions to each branch releases can serve of evidence of emphasis or concern regarding this degradation in performance. At present, by browsing the source control history it is evident that branch 2.x receives the majority of source code commits, and new versions are released constantly. The other two branches, although active, receive much less attention from Hadoop developers.

A. Mining mentions of Hadoop Performance

To determine if the community noticed this change in performance we mined StackOverflow questions that asked about it and the bugtracker for Hadoop Common, Hadoop MapReduce, Hadoop Yarn and Hadoop HDFS.

StackOverflow is a question and answers website that programmers use for general support. Sometimes project specific questions appear on the StackOverflow site. From StackOverflow, we retrieved the database dump and parsed through all of the available questions tagged with `hadoop`, `yarn` and `mapreduce`.

Hadoop performance is a common topic found within StackOverflow posts made by users when deploying a particular release. Most posts regarding Hadoop performance promote the fine-tuning of configurations files, and using the infrastructure characteristics to achieve better performance. Hardware, Operating System, and Java characteristics are commonly addressed as the source of Hadoop performance problems, often overlooking the behaviour of internal framework components.

As we can observe in Figure 2, YARN is not an uncommon topic, especially on the subject of messages. Although the YARN performance was hinted at, it was not specifically addressed by users. Questions regarding general Hadoop performance are more far common, and in general, do not associate YARN issues with Hadoop performance.

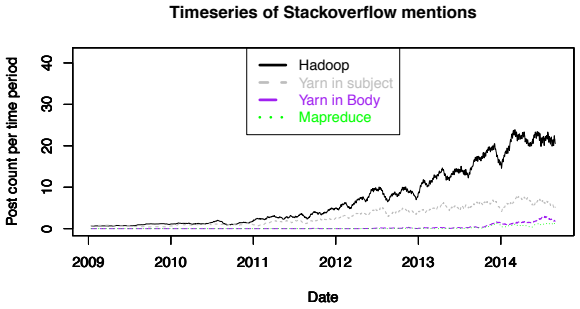

Figure 2. Hadoop mentions on StackOverflow

IV. Methodology

In this study, we tested 12 Hadoop releases from the current development branches to analyze the performance and energy consumption differences. Each release was tested using the Sort (I/O-bound) benchmark, across different dataset sizes, on our 9-node cluster infrastructure. Each Hadoop job was configured to run with the same configuration across the different tested releases, including the MapReduce parameters and number of job tasks. The cluster is also instrumented with 4 *Watts Up? Pro*[4] power meters. We generated datasets with different sizes for the experiments: 10, 48, and 256GB datasets. For the 10GB dataset, the benchmarks ran 10 times for each release/dataset. These experiments ran using the head node and only four DataNodes. For the larger datasets each the benchmarks ran 5 times for each release/dataset using all the 8 DataNodes.

We analyzed three Hadoop development branches to identify reasons for their different performance and energy needs. Initially we mapped the releases of the Hadoop project to develop a genealogy tree of the project. Using the release logs and commit information, we were able to outline the project release dependency, showing which releases were used for each branch. Figure 1 presents the Hadoop development history and the selected releases for our study (the 12 highlighted ones).

Our selection includes releases from 1.x branch (1.0.0, 1.0.3, 1.1.1, and 1.2.1), supported before the YARN development.

[4]http://www.wattsupmeters.com/

The other releases are from 0.23.x (0.23.3, 0.23.6, 0.23.8, and 0.23.10) and 2.x (2.1.0, 2.2.0, 2.3.0, and 2.4.0) branches. These two branches are responsible for major architectural modifications in the project, including the YARN resource manager, which was developed as Hadoop evolved in order to promote flexibility.

V. Energy Performance Results

Our experiments show large differences in energy consumption between multiple tested Hadoop releases. With the evolution in Hadoop branches, releases that included the YARN resource manager performed worse and consumed more energy when compared to the 1.x releases. The differences between the three branches are easily identified. Releases 0.23.x consumed on average 39% more energy than releases 1.x, releases 2.x consumed on average 60% more energy when compared with the 1.x releases. The larger datasets also corroborate the results: the 48GB Sort showed that the energy consumption on 0.23.x and 2.x releases is still higher than releases 1.x, although proportionally lower when compared to the 10GB Sort; the 256GB Sort results show the same pattern again, with releases 1.x saving considerably more when compared to the other releases.

VI. Analysis

To understand what affected Hadoop's performance we investigated the correlation of several factors with the energy consumption such as the software size versus time and energy, and a set of object-oriented metrics.

A. Size and Time versus Power

One confounding factor of any analysis of software and metrics is the size of the software versus metrics and performance. The *Lines of Code* (LOC) will correlate linearly with many features of the source code. Thus we ask, "Does Hadoop size correlate with energy?". We define size as the number of Java lines of code counted by David Wheeler's SLOCCount program for that version of Hadoop. For all tests of all sort tasks, for all Hadoop version (0.23.3, 0.23.6, 0.23.8, 0.23.10, 1.0.0, 1.0.3, 1.1.1, 1.2.1, 2.1.0, 2.2.0 2.3.0 2.4.0) the Pearson correlation is 0.1496, which is a very weak to negligible correlation. The problem is that this measurement mixes tasks of different sizes. Pearson linear correlation values, evaluated per sort task, were between 0.8345 and 0.9018 with a median of 0.8770. We argue that size does indeed correlate with energy, but if we control for other factors, such as version we might find that it is not as important.

On a single computer typically time and energy are related as energy is defined as the integration of power over time ($e = p \cdot t$). Yet the Hadoop task runs on multiple computers, thus we ask "Does energy correlate with Hadoop run-time?" Regardless of the sort task size or configuration, over all tests, a Pearson correlation of 0.9880 was achieved, indicating high linear correlation between time and energy, as expected. Yet when analyzed per sort configuration, described previously, the range of correlation per task varies between 0.5927 to 0.9648 with a median of 0.6797. This implies that in general time is linearly correlated to power, but within a test configuration it has a medium to high strength correlation. Thus time acts as a medium to high accuracy proxy for energy.

The relationship between energy and time is established, and the medium to strong relationship between size and energy is established. Thus we want to know, "Does Hadoop Size correlate with run-time?". Regardless of sort configuration a low Pearson correlation of 0.1240 between Java LOC and run-time was achieved, indicating that code size and run-time were not related. Per each sort configuration the correlation was higher between low correlation of 0.2710 to a strong correlation of 0.7552, with a median correlation of 0.5285. Thus code size had a weak to medium strength relationship depending on the task.

If code size mattered, and code size increased over time, perhaps the release order, the version number, matters. Thus we ask the question, "If we order the Hadoop releases by release date, does the rank of the release (first, second, third, etc.) correlate with energy?" Regardless of the sort configuration, a low Pearson correlation of 0.1133 was measured. When broken up per sort configuration task the correlations were still quite low from 0.0046 to 0.2736. If sorted lexicographicaly, the range of correlations improves from 0.1349 to 0.3611. If ordered from 1.0.0 to 0.23.3 to 2.1.0 to 2.4.0 — 1.0.0, 1.0.3, 1.1.1, 1.2.1, 0.23.3, 0.23.6, 0.23.8, 0.23.10, 2.1.0, 2.2.0, 2.3.0, 2.4.0 — we find a weak general correlation (0.1700) and a strong per sort correlations ranging from 0.7814 to 0.8799. Using this ordering, the Pearson correlation between version and size is 0.9614 in general, and from 0.9572 to 0.9944 correlation per sort configuration. Thus we find that Hadoop versions can correlate heavily with energy consumption, but it depends on how the versions are ranked. Furthermore the date of the Hadoop release is not correlated with energy consumption. This indicates that the branch matters, and so do the design choices in them.

In summary, we found that time and energy had a general medium-strength correlation, while Hadoop version number (i.e., branch), not release date, had a strong correlation with energy consumption. We also found while size was strongly correlated with energy consumption, the version was an equally strong predictor of energy consumption, as both are correlated with each other. Thus we argue that while LOC looks strong, it is the version and revisions to the code which truly affect energy consumption.

B. CKJM Versus Hadoop

Most of Hadoop is written in Java, an Object Oriented language. The CKJM-extended suite [6] measures a variety of object-oriented metrics for Java. These metrics were first proposed by Chidamber and Kemerer [1]. Many of the CKJM metrics we investigate are averaged over all classes.

Some of the metrics we measured are described by Jureczko et al. [6]: *data access metric* (DAM) — a ratio of private and protected attributes to total attributes per class; *Afferent couplings* (Ca) — how many other classes use a specific class; *number of children* (NOC) — the number of direct children of a class; *response for a class* (RFC) the number of possible methods that a class could call; *weighted methods per class* (WMC) — the number of methods per class; *number of public methods* (NPM); *measure of functional abstraction* (MFA) — a ratio of inherited methods over the total number of methods; *Lines of Code* (LOC) CKJM's binary version of LOC, it sums the number of fields, methods and the number of instructions per method in a class; *lack of cohesion of methods* (LCOM and LCOM3) — counts methods who don't share attributes or fields; and finally *cohesion among methods of class* (CAM) —

measures whether or not methods of a class share the same types of parameters or not.

Per each version of Hadoop we ran the CKJM extended suite on the Hadoop java jar files. The bytecode of each class was analyzed and metrics produced. Then we averaged the metrics over the entire product. Once this is done, one can compare the benchmarked energy measurements and time measurements against the extracted CKJM metrics. 1 version of Hadoop will have 1 set of average CKJM metrics.

To investigate if CKJM metrics were related to Hadoop performance we produced thousands of linear models and evaluated them, keeping only 415 of those that met a stringent criteria. This kind of multiple regression analysis is similar to ANOVA. All combinations of CKJM metrics, java LOC, and version (ranked from 1.x to 0.23.x to 2.x) features were evaluated, but if they were not linearly independent (correlation of 0.75 or less) the model was not evaluated. If a model was produced and all independent variables were not significant ($\alpha \leq 0.05$) then the model was not kept.

The linear models per sort configuration were successful as the R^2 range was between 0.9035 and 0.9998 for the top 10 performing models of each configuration where all variables were significant. A linear model of $e \sim NOC + Ca + DAM + version$ was the top model for 48GB Sort and 256GB Sort experiments, while a similar model of $e \sim RFC + Ca + DAM + LOC$ was the top model for 10GB Sort. Version or average size (LOC) appeared in some of the top models, demonstrating the power of size or version awareness.

For each Sort experiment, given the top 10 performing models of each configuration, the number of occurrences of a metric in the top 10 models were counted. Version numbers appear in 4/5 of the task's top 10 models, while size (java LOC) does not appear in the top 10 models of any of the tasks. Another form of size, average binary size (CKJM-extended's LOC metric, not to be confused with source LOC) does appear in many of the top 10s. Yet the most dominant metrics are DAM and NOC followed by Ca, RFC, WMC, NPM, MFA, LOC, LCOM3, LCOM, CAM and version. Out of 50 models, *version* appears in 9 of them while NOC appears in 30/50 and DAM appears in 24/50.

Models consisting solely of NOC and DAM were in the top 10 models of 4/5 jobs. Both NOC and DAM negatively correlate with the number of lines of Java. Furthermore NOC, number of children, likely negatively correlates with the age and evolution of the product as more classes are added, there are less children. If a child class is added, average NOC should decrease. NOC increased from version 1 to the 0.23 versions and then dropped significantly from 0.3074 to 0.2592 for version 2.1.0 and later. DAM was higher for the version 1 branch than for the 0.23 branch and increased again after 2.1.0, but version 1 had higher DAM than later versions. Thus there's no strong correlation between the successful OO metrics, such as DAM or NOC, and the version number or the lines of java code. This indicates that perhaps there is some relationship between the general structure of the code and its final performance in terms of energy, but size and version awareness still play a factor. DAM and NOC were also correlated with energy consumption in prior studies [4].

VII. Conclusions

In this paper, we presented our investigation of the Apache Hadoop project, its branches and releases, regarding energy consumption and performance. Analyzing 4 releases of each of the 3 current branches — 1.x, 0.23.x, and 2.x — we demonstrated through experimentation that there is a significant difference in energy consumption between the framework development branches.

The 1.x releases had better performance and lower energy consumption than all the other tested releases. Releases 1.1.1 and 1.2.1, which did not incorporate the new resource manager of the framework, YARN, demonstrated to be the more energy efficient in the experiments. We acknowledge that the evolution of the framework brought desirable characteristics to the platform, but we argue that the same evolution, through the development of the YARN resource manager, included in 0.23.x and 2.x releases, caused significant loss in performance, and consequently, increases in job energy consumption. As shown in subsection VI-A, the version and release have a strong correlation with the energy consumption in Hadoop.

Therefore, unless users need such features presented in the latter development branches, they should select Hadoop releases from 1.x branch, especially 1.2.1 and later, and apply the compatible patches for upgrades and stability. Furthermore, anecdotal evidence from an analysis of StackOverflow questions about Hadoop support the hypothesis that users are not aware of such new features, and, by extension, the performance and energy penalties they incur.

VIII. Acknowledgements

This work was funded by Fundação Araucária, CNPq-Brazil, and by the Emerging Leaders in the Americas Program (ELAP) from the Government of Canada. Abram Hindle and Denilson Barbosa are supported by NSERC Discovery Grants.

References

[1] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[3] A. E. Hassan. The Road Ahead for Mining Software Repositories. In *Proceedings of the Future of Software Maintenance (FoSM) at the 24th IEEE International Conference on Software Maintenance*, 2008.

[4] A. Hindle. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, pages 1–36, 2013.

[5] A. Hindle, A. Wilson, K. Rasmussen, J. Barlow, J. Campbell, and S. Romansky. GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework. In *Mining Software Repositories (MSR), 2014 11th IEEE W. Conf. on*. ACM, 2014.

[6] M. Jureczko and D. Spinellis. *Using Object-Oriented Design Metrics to Predict Software Defects*, volume Models and Methodology of System Dependability of *Monographs of System Dependability*, pages 69–81. Oficyna Wydawnicza Politechniki Wroclawskiej, Poland, 2010.

[7] D. Li, A. H. Tran, and W. G. J. Halfond. Making web applications more energy efficient for oled smartphones. In *Proceedings of the International Conference on Software Engineering (ICSE)*, June 2014.

[8] I. Polato, R. Ré, A. Goldman, and F. Kon. A comprehensive view of hadoop research – a systematic literature review. *Journal of Network and Computer Applications*, 46(0):1 – 25, 2014.

[9] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. N. Nasser, and P. Flora. An exploratory study of the evolution of communicated information about the execution of large software systems. In *WCRE*, pages 335–344, 2011.