Double Dip Map-Reduce for Processing Cross Validation Jobs

Danilo Moret
Webmedia
Globo.com
moret@corp.globo.com

Karin Breitman, Evelin Amorim, Jose Talavera, Ruy Milidiu
PUC-Rio
{karin,eamorim,jherrera,milidiu}@inf.puc-rio.br

Jose Viterbo
Instituto de Computação
Universidade Federal Fluminense
viterbo@ic.uff.br

ABSTRACT
Cross validation is fundamental to machine learning as it provides a reliable way in which to evaluate algorithms and the overall quality of the corpora in use. In typical cross validation, the corpus is initially divided into *learning* and *training* segments, then crossed-over in successive rounds, so that each data segment is validated against the remaining ones. This process is prohibitively time and effort consuming, and often brushed off for computationally cheaper ones, such as heuristics. In this paper we introduce a cloud-based architecture for running cross validation jobs. Our solution makes heavy use of computational resources in the cloud by proposing a strategy in which there are two distinct, subsequent, map-reduce cycles: the first to perform the algorithmic target computation, and the second to provide cross validation data to retrofit the machine learning process. We demonstrate the feasibility of the proposed approach, with the implementation of a web segmentation algorithm.

Categories and Subject Descriptors
C.2.4 [**Cloud Computing**]

General Terms
Algorithms, Design, Experimentation.

Keywords
cloud computing, cross validation, k-fold, machine learning, map-reduce, aws

1. INTRODUCTION
The map-reduce technique is a tried and true way to enable large parallel processing [1]. Plenty of problems can be addressed in a distributed way by first splitting the input data into independent chunks, processing them in separate nodes, and finally merging the results. The cross validation pre-processing stage for machine learning is a perfect example of this class of problem. According to Refaeilzadeh et al [2] cross validation is a statistical method useful in evaluating and comparing two or more learning algorithms, by dividing a corpus of data into training and leaning chunks. In traditional cross validation, chunks are crossed-over in successive rounds and validated against one another.

The cross validation technique can be easily restated in terms of a map-reduce solution, i.e., split a large corpus into separate chunks, process each chunk individually, analyze the results. It is interesting to note that the analysis step, in turn, can also be stated as a map-reduce job. This means that an efficient, cloud based, implementation of the cross validation technique requires double use of the map-reduce paradigm, hence the paper title.

In this paper we propose a cloud-based solution for handling large cross validation jobs. We illustrate our approach with an implementation of area segmentation algorithm, whose goal is to parse web pages into different segments (areas) that correspond to different types of information. The algorithm used on the test is the one devised by Amorim et al [3], which has a mixed approach that uses some features from the DOM tree to segment web pages.

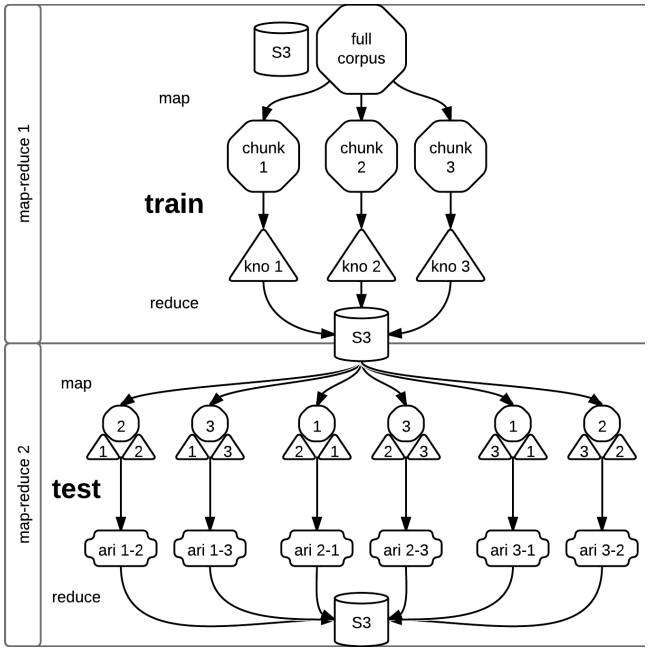

Figure 1: Double dip map-reduce

2. PROPOSED ARCHITECTURE
In the proposed solution we apply the map-reduce technique on two distinct, subsequent phases, as illustrated in figure 1. During the first phase we split the corpus on n chunks, and map a *train* job to process each individual chunk. The jobs are accessible by a pool of workers via a queue. Each job consists of downloading chunks of the corpus stored on a storage service, uncompress it to the appropriate folder, run the script, collect the output, and upload the results back to the storage service, as illustrated by the map-reduce1 box, in figure 1.

During the second phase, the mapping step is responsible for combining each result (embodied by a *kno* file, that contains the knowledge model generated by the *training* algorithm used in the previous phase) with all other available *kno* type files, producing

SAC'12, March 26-30, 2012, Riva del Garda, Italy.

n-1 *training* jobs for each of the n *kno* chunks, for a total of n * (n - 1) jobs. In the example in figure 1, we have 3 chunks, resulting in 6 test jobs.

These second phase jobs are also accessed by a pool of workers. Each one of the working nodes of our architecture downloads two *kno* files, as well as the corpus chunk needed by the method being executed. The node then runs the *test* method script, computes the results (embodied by a floating point number, that provides a measure of the accuracy of the computation, the *ari* value) and uploads it back to the storage service, as illustrated by map-reduce 2 box in figure 1.

Our two phase strategy can be thus summarized: on phase 1 we split the corpus on manageable chunks and run the map-reduce implementation to *train* all chunks. On phase 2 we *test* all *knos* against the other *kno*'s and chunks. Finally all *ari* numbers are computed.

3. INITIAL DESIGN AND ESTIMATES

3.1 System concept and design

The cross-validation required by the technique proposed by Amorim et al takes into consideration the whole corpus. As per the process proposed in the last section, the first step consists of splitting the corpus into several chunks. The following step is *training* each chunk using the *train* method of the proposed algorithm, whose result is the production of a set of segmentation rules for each individual chunk.

The rule sets - embodied in *kno* files - are then used for the cross-validation step. Cross validation is done by running the *test* method of the algorithm. The result of each execution is a metric that serves to evaluate the results, a number called *ari*.

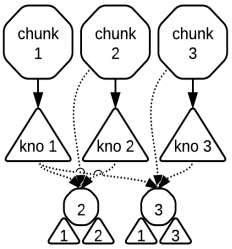

Figure 2: Training phase detail

The algorithm implementation of those two methods - *train* and *test* - are responsible for performing the actual machine learning tasks as proposed by Amorim et al [3]. It's a combination of Python scripts and binaries, making use of C4.5 binaries implemented and compiled by Quinlan [4]. The authors also provide four corpora, that combined, exceed 30GB of uncompressed html files. This volume of data makes it impossible to compute the cross validation for the combined corpus directly in a traditional, sequential manner. See predictions in table 1 of the total time to execute on a single machine.

In the proposed cloud-based solution we use the implementation proposed by Amorim et al of their algorithm as a "black box appliance", with two interaction points: the first as a script to execute the *training* method, that takes a folder containing the uncompressed chunks of the combined corpus, and produces the *kno* file. The second interaction point was an additional script to run the *test* method, that takes as parameters a folder containing the uncompressed chunks to be compared and, additionally, two *kno* files, one for the trained chunk and another for the chunk to be compared, as illustrated in figure 2.

3.2 Estimates

To evaluate the financial viability of the proposed solution we computed a few scenarios. We based our estimates on running times measured on local runs of the *training* and *test* scripts.

Our times on local experiments were averaging around 60s with the *training* script and 1.5s with the *testing* script. The sample corpus chunk used was of 5MB uncompressed. Assuming that time would grow linearly as the size of the corpus grew we extrapolated those values to estimate the amount of processing time and cost in USD of the experiment.

Assuming the following:

corpus size: S = 30GB; piece size: p = 50MB
piece training time for 50MB: t50 = 600s
piece testing time for 50MB: c50 = 15s
mac.h (machine hours) cost: d = 0.17USD/h

We obtained:

# of pieces: n = S / p = 600;
training total time: T = n*t50 = 100mac.h
testing total time: C = n*(n - 1)*c50 = 1498mac.h
total cost: D = (100 + 1498)*0.17 = 271.66USD

4. TECHNOLOGY AND EXECUTION

We decided to embrace Amazon Web Services (AWS) as our cloud computing platform. Several factors contributed to our decision. To cater to our storage needs we used Amazon's S3 [5]. The Linux environment was chosen to match the machine learning development environment. We picked a clean Ubuntu Server Amazon's Machine Image (AMI) as the starting point, installed tools needed and stored the modified image following documented instructions. We chose CloudCrowd as our distributed processes manager [6]. It handles both the job queue and scheduling the workers to pool it.

4.1 Initial run

We took a sample of twenty chunks of the corpus to run experiments on the system deployed. We used only five instances of varying types - some small, some medium - to measure performance, stability and usability of the system.

The initial executions raised concerns about performance of the proposed solution. One of the assumptions we made earlier was that *training* time would increase linearly, which proved to be wrong. Also the average time we had running locally was about a fifth the average time on a medium instance. To aggravate this situation, the average processing time varied greatly from one chunk (job) to another.

System stability was good and very few glitches showed up on the cloud environment. However, an issue that required improvement was the ability to stop and resume the execution without reworking previously trained pieces or cross-validated pairs.

To deal with the performance issues we resourced to re-planing and re-estimating, as detailed on the next session. The solution to the few glitches encountered, as well as the need to resume execution, were implemented separately and retrofitted into the code.

4.2 Reestimation and second run

Each 50MB piece was taking far more time than the original sample piece of 5MB in the *training* phase. Instead of a linear 10x increase, or even an exponential 100x, it was noted to take even more time. The performance on the *testing* phase was as expected,

but given the great variation on each pair we decided to consider it lower than the initial estimate, for safety.

Given the performance results, not only the costs would greatly surpass the available budget, but also take several days to execute. To deal with that we had two options. First, we could split the corpus in smaller chunks. However an increase of 10x of chunks meant increasing the amount of *testing* pairs 100x. Second, we could use a smaller portion of the corpus, processing less GBs this time and attempting a complete run later.

We decided to use the first option. But before ruling the second option out, we drew estimates varying the corpus size to compare costs. Interestingly enough, the number of instances involved did not make the final cost vary a great deal, as the amount of machine hours needed is constant and the only cost that increased when using less instances was the cost of the small server node instance. But because the final time to run the experiment augmented as a result, so did the overall cost. Note that computational resources in AWS are charged by the hour.

However, as we implemented a resume feature, with a simple modification we could reconfigure the computation to take into consideration only a smaller fraction of the corpus, and go for larger fractions later. That was the new plan: we would start with small fractions and increase as long as we had time on our schedule. Our target was getting 500MB of the corpus processed.

After splitting the corpus and re-uploading the pieces to S3 the system was set up to work on 100 pieces. The execution took place on 19 instances, and ended after three hours. We achieved relatively good results, but hit on new problems. This was the first time that some parts of the corpus were sent for *training*. Given the nature of the ETL scripts, they depend on overall corpus consistency, in this embodied by an uniform encoding, preferably UTF-8. Unfortunately the corpus was not properly encoded, and about a quarter of the units failed to *train*.

The second problem found was more severe. All previous executions had to list from S3 well under a thousand entries. Because of that we overlooked a limitation on the S3 API that affected the AWS::S3 library, which in turn affected the execution time.

In our implementation the mapping step of the second phase listed the results of the first phase - the *kno* files - which were stored together at the same bucket on S3 that the pieces. But because of the API limit, the system was only able to access the *kno* files that were among the first thousand files when listed in alphabetical order. So, after the execution ended, we found only a fifth of the expected *testing* results - the *ari* values.

We corrected each of those problems and rerun the experiment subsequently, finally obtaining clean results. Final execution times are listed on table 1.

Table 1. Total execution times and predictions (in italics) for multiple instances

	Corpus Size					
Instances	**5 MB**	**100 MB**	**200 MB**	**500 MB**	**5000 MB**	**30000 MB**
1	0.5h	*10h*	*20h*	*50h*	*500h*	*3000h*
5	n/a	3.12h	*7h*	*20h*	*200h*	*1200h*
20	n/a	0.86h	2h	5.62h	*57h*	*350h*

5. CONCLUSION

Traditionally limited by the available computer infrastructure, machine learning experiments, in particular those using cross validation techniques, are prohibitively time and effort consuming. The availability of public cloud computing services providers offers access to highly available computing resources at reasonable pricing. In this paper we describe a cloud-based architecture for running cross validation jobs. Our solution makes heavy use of computational resources by proposing a strategy in which there are two distinct, subsequent, map-reduce cycles: the first to perform the algorithm target computation and the second to provide cross validation data to retrofit the machine learning process.

The proposed architecture generates feedback to the machine learning algorithm directly. Amorim et al made improvements to the algorithm to collect other metrics as well. Future experiments will take additional metrics into consideration. We also plan to run additional experiments with larger corpora, to evaluation the scalability of the proposed solution.

We also received suggestions to run a series of experiments using public corpora, to facilitate the update of results, benchmark with other solutions and allow for independent reruns. On the next experiment we aim at using a corpus that is both meaningful to visual segmentation and publicly available. We are currently working on an interface that will allow web service access to the the proposed architecture.

6. REFERENCES

[1] Dean, Jeffrey and Ghemawat, Sanjay; MapReduce: Simplified Data Processing on Large Clusters, Google Inc., OSDI 2004

[2] Refaeilzadeh, Payam; Tang, Lei; Li, Huan; Encyclopedia of Database Systems, pp. 532-538 - Arizona State University, 2009

[3] Laber, Eduardo S., Souza, Críston P. de , Jabour, Iam V., Amorim, Evelin C. F de, Cardoso, Eduardo T., Rentería, Raúl P.,Tinoco, Lúcio C., Valentim, Caio D.: A fast and simple method for extracting relevant content from news webpages. CIKM 2009: 1685-1688

[4] Quinlan, J. Ross: C4.5 Programs for Machine Learning, Morgan Kaufmann Publishers, 1993

[5] Amazon Simple Storage Service (S3) - http://aws.amazon.com/s3/ - Amazon

[6] Ashkenas, Jeremy; CloudCrowd - https://github.com/documentcloud/cloud-crowd/wiki - The New York Times & DocumentCloud

Acknowledgment

The authors thank the Brazilian Institute for Web Science Research (CNPq 557.128/2009-9 and FAPERJ E-26/170028/2008) and Globo.com for the support to this work.