ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

Communication Challenges in Cloud K-means

Matthieu Durut[1] and Fabrice Rossi[2]

1- Lokad, 70, rue Lemercier, 75018 Paris – France

2- BILab, Télécom ParisTech, LTCI - UMR CNRS 5141
46, rue Barrault, 75013 Paris – France

Abstract. This paper studies how parallel machine learning algorithms can be implemented on top of Microsoft Windows Azure cloud computing platform. More specifically, we design efficient storage based communication mechanisms that lead to a scalable implementation of the K-means.

1 Introduction

Large scale datasets are becoming increasingly common [4] and as a consequence, parallel data mining and machine learning algorithms are needed. Even with such algorithms, massive computing resources are needed to handle the largest datasets. Unfortunately, the cost of ownership of such resources is as large as they are, while the computing paradigms of supercomputer is difficult to master. As a consequence a new class of services has appeared under the umbrella name of Cloud computing [1]: they allow customers to rent very large computing resources (generally up to 1000 CPU cores) on demand on a hourly basis, while providing rather simple computing API. We study in this paper how the machine learning computing can leverage one of those cloud solutions, Microsoft Windows Azure, to implement large scale data analysis.

2 Parallel K-means

We use the k-means clustering algorithm as a typical example of machine learning methods. This choice is mainly motivated by the fact that apart for the number of iterations to convergence, the processing time of k-means is known before running, and only depends on the data dimensions and on K, rather than on the actual data values: timing results obtained on simulated data then apply to any data set with the same dimensions. k-means has already been successfully parallelized on shared memory computers and on local clusters of workstations: numerous publications (see e.g., [3]) report linear speedup up to at least 16 processing units (PU, which can be CPU cores or workstations).

Let us recall the principles of Dhillon and Modha's parallel k-means [3]. k-means consists in alternating two phases (after proper initialization). In the assignment phase, the algorithm computes the Euclidean distance between each of the N data points $X_i \in \mathbb{R}^D$ and each of the K cluster prototypes $m_j \in \mathbb{R}^D$. Each X_i is assigned to its closest prototype, denoted m_{k_i}. In the recalculation phase, each prototype is recomputed as the average of the data points assigned to it in the previous phase. Since distance calculations are intrinsically parallel, it

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

is therefore natural to split the computational load by allocating disjoint subsets of N/P data points to P PU. First, all the PU - called mappers following [2] terminology - complete their assignment phase and compute their local prototypes version. Second, reducers merge the local prototypes versions into a new shared version, which is broadcasted back to all the mappers. In [3] this is done through a dedicated efficient "reducing" function of the MPI library which uses $\log P$ parallel rounds of communications between the PU. This results in a cost of $O(KDlog(P))$.

The cloud implementation studied in this paper is based on the parallel k-means described above. The main difficulty consists in implementing synchronization and communication between the PU that will not affect performances significantly, using the facilities provided by Windows Azure cloud operating system.

3 Microsoft Windows Azure

Windows Azure Platform is Microsoft's cloud computing solution, in the form of Platform as a Service (PaaS). The underlying cloud operating system (Windows Azure) provides services hosting and scalability. It is composed of a storage system (that includes Blob Storage and Queue Storage) and of a processing system (that includes *web roles* and *worker roles*).

Web roles are designed for web application programming and do not concern the present article, while worker roles are designed to run general background processing. Each worker role typically gathers several *cloud services* and uses many *workers* (Azure's processing units) to execute them. Our implementation uses only one worker role, several services and tens of PU.

Azure storage system is used to implement synchronization and communication between workers. It must be noted indeed that Azure does not offer currently any standard API for distributed computation, neither a low level one such as MPI, nor a more high level one such as Map Reduce [2] or Dryad [5]. Map reduce could be implemented using Azure components (following the strategy of [6]), yet, as pointed out in e.g. [7], those high level API might be inappropriate for iterative machine learning algorithms such as the k-means. We rely therefore directly on Azure queues and blob storage.

Azure Queues provide a message delivery mechanism through distributed queues. Queues are designed to store a large amount of small job messages. Using queues to communicate helps building loosely coupled components and mitigates the impact of individual component failure. Messages stored in a queue are guaranteed to be returned at least once, but possibly several times: this requires one to design idempotent jobs. If a worker fails to complete a job (because it throws some exception or because the worker dies), the message is requeued after a certain period of time. Through this process, one can make sure no job is lost because of e.g., a hardware failure.

Azure Blob Storage enables applications to store large objects, up to 50 GB each. Blobs are composed of a string (that is used as a key to store the value), of

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

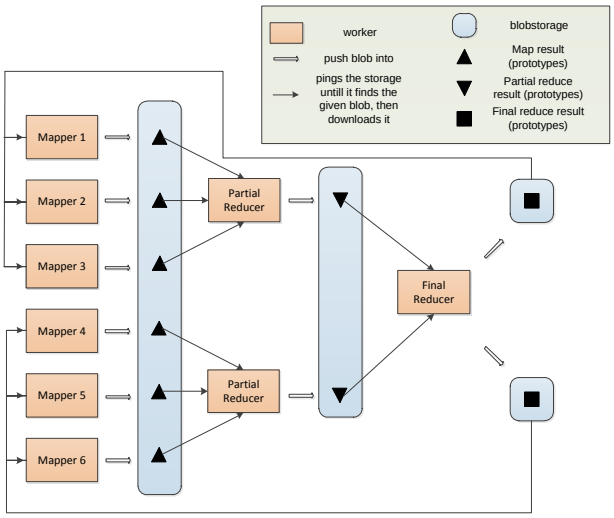


Fig. 1: Workers communication architecture.

a value storing a binary object, and of a timestamp (etag) that indicates the last write on this blob. In addition of Get and Put methods, blobs whose key shares a given prefix can be listed. Optimistic non locking atomic read-modify-write operations can be implemented using a timestamp matching condition: a write succeed if and only if the timestamp of the storage matches the one provided by the write operation.

Our implementation uses *Lokad-Cloud*[1], an open-source framework providing a small abstraction layer to ease Azure workers startup and life cycle management, and storage access.

4 Proposed implementation

Our implementation consists in three cloud services: setup, map and reduce services (the last two are shown on Figure 1). A queue is associated to each service: it contains messages specifying the storage location of the data needed for the jobs. Workers regularly ping the queues to acquire a message. Once it has acquired a message, a worker starts running the service related to the queue where the message was stored, and the message becomes invisible until the job is completed or timeouts. Overall, we use $P + \sqrt{P} + 1$ processing units in the services described below.

The **Setup Service** first generates the original shared prototypes and pushes them on $\lceil \sqrt{P} \rceil$ different places in the BlobStorage (this way, we avoid contention while all mappers are trying to read on these shared prototypes). Setup Service also generates P split data sets of N/P points each and also put them into the BlobStorage. Once completed, it launches Map Service.

[1] `http://code.google.com/p/lokad-cloud/`

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

When executing a **Map Service** job, a worker first downloads the data set it is in charged of (once for all). Then the mapper loads one of the copy of the initial shared prototypes and starts the computation step that consists in attributing each point in the data set to the closest prototype. As the points are being processed, the worker build on the fly its new local prototypes version. When completed, the mapper pushes its local prototypes version into the storage in accordance with the following addressing rule: iteration/groupId/jobId.

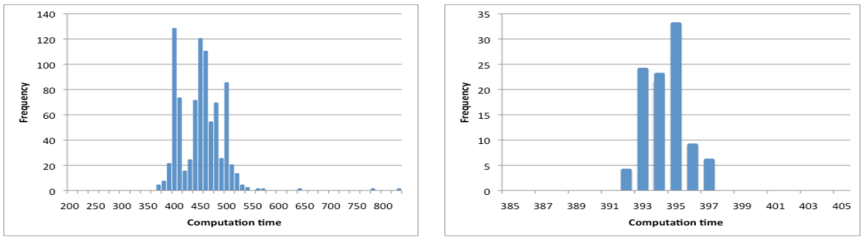
Each **Reduce Service** worker is in charged of listing and loading as soon as they become available the map results within a given iteration/groupId directory until it gets all the expected results within this directory. The reducer then merges the different prototypes versions, and pushes the merged result into the storage in the partial reduce directory. One last reducer runs the same process on the partial reduce results directory. Once the $\lceil\sqrt{P}\rceil$ partial reduce results are retrieved, the last reducer builds a new shared version out of the partial reduce results and pushes a copy of it into each of the $\lceil\sqrt{P}\rceil$ places in the shared prototypes directory. After several seconds, every mapper has been pinging and loading one copy of the now available shared result blobs, and the map step can be run again.

5 Performance analysis

We designed several experiments to analysis the performances of the proposed implementation. We first study straggler issues, then communication performances, then the actual performances of the algorithm.

5.1 Straggler issues and impact on performances

We first set an experiment with 85 mappers, each mapper performing 10 times a 7 minutes task. No communications were used between workers here, we only recorded the empirical distribution of the computation time of the 850 tasks run, as reported in Figure 2(a).



(a) Empirical computation time distribution

(b) Single worker empirical computation time distribution.

Fig. 2: Empirical distributions.

Results contain outliers that correspond to temporary slow processing: the three slowest runs (823 seconds, 632 s, 778 s) have been performed by a virtual

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

machine (VM), which has also performed very well on other iterations (e.g., 360 s). This could be explained by the fact the physical machine hosting our VM had been hosting temporarily another VM.

Additionally, Figure 2(a) shows a 3 empirical modes in the main interval (from 390 s to 500 s, covering 90 % of the runs). Except for the single (virtual) machine with the longest runs, each machine performed the 10 runs in only one of the 3 modes: for instance, Figure 2(b) reports 100 iterations of 7 minutes on a single machine. Therefore, the 3 modes may be due to hardware heterogeneity or multiple VM hosted on the same physical machine.

Those so-called **straggler issues** have already been observed, for example in the original MapReduce article [2] by Google, but only while running thousands of machines. We show that straggler issues are also observed on a small pool of workers such as 100 VM. Authors of [2] describe a monitoring framework to detect tasks taking too much time, and use backup workers to relaunch tasks that has been detected to be too long. But, this approach implies to wait for the standard duration of the task before detecting straggler tasks and launching again the corresponding jobs. Therefore, for a given number of PU P, using [2] approach on straggler issues leads to speedups of no more than $\frac{P}{2}$. In the context of high performance machine learning, this maximal speedup seems low (see [7] for other examples of limitations of MapReduce for machine learning).

5.2 Communication benchmark

We fixed K=1000, D=1000 (each prototypes version size is therefore 8MB) and I=10. In order to record communication time without being affected by straggler issues (as reported above), we run our clustering implementation for different values of P, replacing the processing part by waiting a fixed period of time (15 seconds). The following table displays wall time (WT) of the fake clustering for 10 iterations, and the amount of time spent in communication (WT - 10 times 15 seconds).

P	5	10	20	30	40	50	60	70	80	90	100	110	120
WT (in s)	287	300	335	359	392	421	434	468	479	509	533	697	591
Comm. (in s)	137	150	185	209	242	271	284	318	329	359	383	547	441

As one can see, communication is not free, especially for small values of P. Yet, communication costs does not move much as the number of worker grows. This is because we design our communication algorithm to limit this. Most of communication costs are due to aggregated read bandwidth boundaries while all workers are trying to read one copy of the shared prototypes at the same time, the final reducer write bandwidth boundary (this worker has to push into the storage $\sqrt{P}$ copies of the shared prototypes in parallel), queues latencies, and some blob temporary unavailability: while the storage is stressed, a given blob already pushed can take as much as 1 minute before being available to download.

5.3 Scaleup results

Instead of trying to optimize P for a given value of (N, D, K) we set the number of mappers to P, and try to provide the best speedup possible, namely P. In the

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

following experiments, we set K=1000, D=1000, and set N to 5 10^4 P. Thus, each worker will be working on 50,000 points, which represent an important part of the RAM of the VM (around 400 MB), helping to achieve the best speedups possible by giving heavy computation loads on each worker. The algorithm is run for 10 iterations to get stable timing estimates (more iterations would be run on a real k-means on such a heavy data set). Sequential time refers to the time that would be spent to run the clustering on a unique worker with infinite RAM and computation speed equals to the median computation speed recorded in 2(a).

P	10	20	40	60	80	100	120
N	5 10^5	10 10^5	20 10^5	30 10^5	40 10^5	50 10^5	60 10^5
Wall Time (in s)	2160	2478	2385	2642	2728	2803	2906
Sequential Time	22,300	44,600	89,200	133,800	178,400	223,000	267,000
Speedup	10.32	17.99	37.40	50.64	65.40	79.56	91.88

6 Conclusion

As reported in [6], cloud storage abstractions are helpful for the design of cloud machine learning applications. Provided that contention is prevented on each storage element and that each worker is granted enough workload, communication becomes small compared to computation, and satisfactory speedups and scaleups can be reached.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

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

[3] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 245–260, London, UK, 2000. Springer-Verlag.

[4] T. Hey and A. Trefethen. *The Data Deluge: An e-Science Perspective*, pages 809–824. John Wiley & Sons, Ltd, 2003.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, New York, USA, 2007. ACM.

[6] H. Liu and D. Orban. Cloud mapreduce: a mapreduce implementation on top of a cloud operating system. Technical report, Accenture Technology Labs, 2009. `http://code.google.com/p/cloudmapreduce/`.

[7] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, California, July 2010.