Elastic CoCoA: Scaling In to Improve Convergence

Michael Kaufmann[†‡], **Thomas Parnell**[†], **Kornilios Kourtis**[†]
[†]IBM Research, [‡]Karlsruhe Institute of Technology (KIT)
[†]Zurich, Switzerland [‡]Karlsruhe, Germany
{kau,tpa,kou}@zurich.ibm.com

Abstract

In this paper we experimentally analyze the convergence behavior of CoCoA and show, that the number of workers required to achieve the highest convergence rate at any point in time, changes over the course of the training. Based on this observation, we build Chicle, an elastic framework that dynamically adjusts the number of workers based on *feedback from the training algorithm*, in order to select the number of workers that results in the highest convergence rate. In our evaluation of 6 datasets, we show that Chicle is able to accelerate the time-to-accuracy by a factor of up to $5.96\times$ compared to the best static setting, while being robust enough to find an *optimal or near-optimal setting automatically* in most cases.

1 Introduction

As data has become a major source of insight, machine learning (ML) has become a dominant workload in many (public and private) cloud environments. Ever-increasing collection of data further drives development of efficient algorithms and systems for distributed ML [8, 2] as resource demands often exceed the capacity of single nodes. However, distributed execution, and the usage of cloud resources, pose additional challenges in terms of efficient and flexible resource utilization. Recently, several works have aim to improve resource utilization and flexibility of ML applications [3, 6, 11].

In this paper, we focus on **Co**mmunication-efficient distributed dual **Co**ordinate **A**scent (CoCoA) [8], a state-of-the-art framework for efficient, distributed training of generalized linear models (GLMs). CoCoA significantly outperforms other distributed methods, such as mini-batch versions of stochastic gradient descent (SGD) and stochastic dual coordinate ascent (SDCA) by minimizing the amount of communication necessary between training steps.

Our work is motivated by two characteristics of the CoCoA algorithm. First, even assuming perfect scalability and no overheads, increasing the number of workers K does not, in general, reduce the time to reach a solution. This is because the convergence rate of CoCoA degrades as K increases [4]. Overall, CoCoA execution is split into epochs, and increasing K reduces the execution time of each epoch, but also decreases the *per epoch* convergence rate, requiring more epochs to reach a solution. Finding the K that minimizes execution time is not trivial and depends on the dataset.

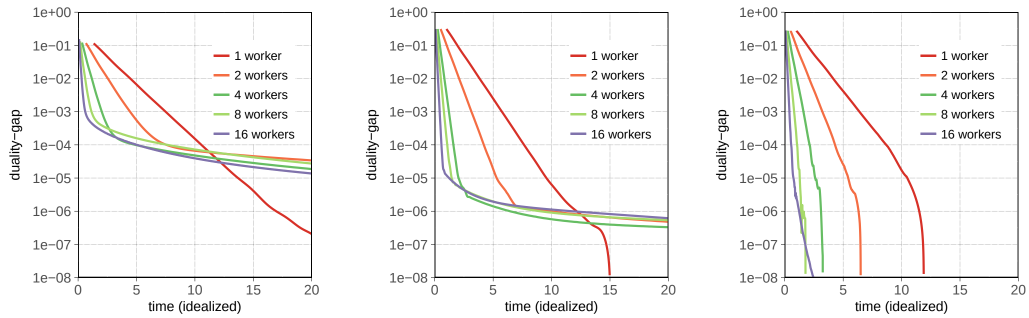
Second, the number of workers K that minimize execution time changes as the algorithm progresses. Figure 1a/1b shows the convergence rate with $K = \{1, 2, 4, 8, 16\}$ workers, using the kdda and higgs datasets as examples. We evaluate the convergence rate by plotting the duality-gap, which is given by the distance between the primal and dual formulation of the training objective, and has been shown to provide a robust certificate of convergence [1, 8]. Both examples show that for larger values of K, the duality-gap converges faster initially, but slows down earlier than for smaller values of K, thus resulting in smaller values for K leading to a shorter time-to-(high)-accuracy[1] than large values for

[1]When we refer to the training accuracy we mean that a highly accurate solution to the optimization problem has been found (i.e., a small value of the duality gap), rather than the classification accuracy of the resulting classifier.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

K. However, this is not universally true, as Figure 1c shows for the rcv1 dataset, which scales almost perfectly with K.

Based on these observations, we built Chicle, an elastic distributed machine learning framework, based on CoCoA, that reduces time time-to-accuracy, robustly finds (near-)optimal settings automatically and optimizes resource usage by exploiting the drifting of the optimal K.



(a) KDDA (b) Higgs (c) RCV1

Figure 1: Example of the convergence of the duality-gap (a certificate for accuracy) for 3 datasets using 1 to 16 workers, assuming perfect scaling and zero communication cost.

2 Background

CoCoA [8] is a distributed machine learning framework to train GLMs across K workers. The training data matrix A is partitioned column-wise across all workers and processed by local optimizers that independently apply updates to a shared vector v, which is synchronized periodically. In contrast to the mini-batch approach, local optimizers apply intermediate updates directly to their local version of the shared vector v, thus benefiting from previous updates within the same epoch.

Due to the immediate local updates to v by local optimizers, CoCoA outperforms previous state-of-the-art mini-batch versions of SGD and SDCA. However, for the same reason, it is not trivial to efficiently scale-out CoCoA, as increasing the number of workers does not guarantee a decrease in time-to-accuracy, even assuming perfect linear scaling and zero communication costs between epochs. The reason for this counter-intuitive behavior is that, as each local optimizer gets a smaller partition of A, i.e. as it sees a *smaller picture* of the entire problem, the number of identifiable correlations within each partition decreases as well, thus leaving more correlations to be identified across partitions, which is slower due to infrequent synchronization steps.

Moreover, as indicated in the previous section, there is no K for which the convergence rate is maximal at all times. This poses a challenge about the selection of the best K. It is up to the user to decide in advance whether to train quickly to a low accuracy and wait longer to reach a high accuracy or vice versa. A wrong decision can lead to longer training times and wasted resources as well as money, as resources – at least in cloud offerings – are typically billed by the hour.

Ideally, the system would automatically and dynamically select K, such that the convergence rate is maximal at any point in time, in order minimize training time and resource waste. As Figure 1b shows, the convergence rate, i.e. the slope of the curve, starting from the same level of accuracy, differs between different settings for K. E.g, as the curve for $K = 16$ flattens when reaching $\approx 1e - 5$, the curves for $K \leq 8$ become relatively steeper until they too, one by one, flatten out. Hence, in order to stay within a region of fast convergence for as long as possible, the system should switch to a smaller K, once the curve for the current K starts to flatten. We assume that the convergence rate, when switching from K to $K' < K$ workers, at a certain level of accuracy, will follow a similar trajectory, as if the training had reached said level of accuracy starting with K' workers in the first place. However, the validity of this assumption is obvious, given that the learned models in both cases are not guaranteed to be indentical.

Apart from the algorithmic side, adjusting K also poses very practical challenges on the system side. Every change in K incurs a transfer of potentially several gigabytes of training data between nodes –

a task that overwhelms many systems [10, 9, 7] as data (de-)serialization and transfer can be very time consuming[2]. It is therefore crucial that the the overhead, introduced by the adjustment of K, is small, such that a net benefit can be realized.

3 Chicle

Chicle[3] is a distributed, auto-elastic machine learning system based on the state-of-the-art CoCoA [8] framework that enables efficient ML training with minimized time-to-accuracy and optimized resource usage. The core concept of Chicle is to reduce the number of workers (and therefore training data partitions), starting from a set maximum number, dynamically, based on feedback from the training algorithm. This is rooted in the observation of a *knee* in the convergence rate, after which the convergence slows down significantly, **and** that this *knee* typically occurs at a lower duality-gap for fewer workers compared to more workers. This can be observed in Figure 1b. Here, the *knee* occurs at $\approx 1e-5$ for 16 workers and $\approx 1e-6$ for 2 workers. The reasoning for adjusting the number of workers is the assumption that CoCoA can be accelerated, if, by reducing the number of workers, it can stay before the *knee* for as long as possible.

3.1 Overview

Chicle implements a master/slave design in which a central driver (master) coordinates one or more workers (slaves), each running on a separate node. Driver and worker communicate via a custom remote procedure call (RPC) framework based on remote direct memory access (RDMA) to enable fast data transfer with minimal overhead. Chicle is implemented in $\approx$ 3,000 lines of C++ code, including the RDMA-based RPC subsystem.

The **driver** is responsible for loading, partitioning and distribution the training data, hence no shared file system is required to store the training data. It partitions the data into $P \geq K$ partitions for K workers, such that each worker is assigned $\frac{P}{K}$ partitions with P being the least common multiple of K and all potential scale-in sizes $K' < K$. Moreover, the central CoCoA component is implemented as driver module. The **workers** implement an SDCA optimizer. Each optimizer instance works on all partitions assigned to a worker, such that it can train with a *bigger picture* once partitions get reassigned to a smaller set of workers. For each epoch, workers compute the partial primal and dual objective for their assigned partitions, which are sent to the driver where the duality-gap is computed and passed to the scale-in policy module.

3.2 Scale-in

Chicle enables efficient adjustment of the number of workers K (and the corresponding number of data partitions per worker process) using a decision policy and a RDMA-based data copy mechanism. In the context of this paper, Chicle only scales-in, i.e. reduces the number of workers K and therefore redistributs the number of partitions P across fewer workers.

Scale-in policy. Our scale-in policy attempts to determine the earliest point in time when it is beneficial to reduce the number of workers K (i.e. the beginning of the *knee*) while, at the same time, being robust against occasional outlier (i.e. exceptionally long) epochs. To that end, we use the slope of the duality-gap over time to identify the *knee*. It computes two slopes (see Figure 2) – a long-term slope S_l which considers the convergence of the duality-gap since the last scale-in event – and a short-term slope S_s, which considers only the last N epochs. As soon as $S_s \times d < S_l$ the policy directs the driver process to initiate the scale-in mechanism. Larger values for N and d generally lead to a more robust decision w.r.t. occasional outlier epochs, however they also increase the decision latency, thus potentially failing to maximize benefits from an earlier scale-in. Empirically, we have determined that $N = 2$ and $d = 1.25$ works well across all evaluated datasets. Our policy does not determine the optimal factor m of the scale-in, i.e. $K \rightarrow K/m$. We use a fixed m of 4, as test have shown that the convergence rate difference for smaller m is often very small.

[2]Initially, we attempted to implement the concept of Chicle in Spark. This, however, failed to a large degree due to very time-consuming (de-)serialization of the training data.

[3]Chicle is the Mexican-Spanish word for latex from the sapodilla tree that is used as basis for chewing gum.

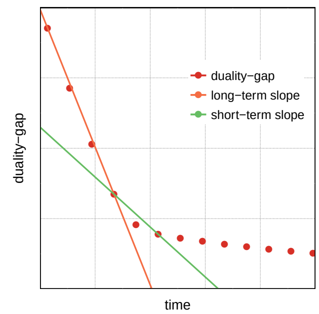


Figure 2: Schematic of the long-/short-term slope of the duality-gap that we use to identify the *knee*.

Scale-in mechanism. We implement a simple, RDMA-based foreground data-copy mechanism to copy data from to-be-removed workers to remaining workers. As the data transfer occurs in parallel, between multiple pairs of workers, we are able to exceed the maximal single-link bandwidth. For a scale-in from K to K/m workers and a single-link bandwidth of r (e.g. 10 Gb/s), we can achieve a total transfer rate of $m \times r$, e.g. 40 Gb/s to scale-in from 16 to 4 workers on a 10 Gb/s network.

3.3 Data partitioning and in-memory representation

While we do not employ a sophisticated data partitioning scheme – we simply split the data into equally sized chunks as it is laid out in the input file – we use an in-memory layout optimized for efficient local access as well as efficient data transfer between workers (see Listing 1). In Chicle, data for each partition is stored consecutively in the `Partition::data` array, which eliminates the need for costly serialization. On the receiving side, a simple deserialization step is required to restore the `Example::dp` pointer into the `Partition::data` array for each `Example`. This data layout, combined with the usage of RDMA, enables us to transfer data at a rate close to the hardware limit.

```
struct Datapoint { uint32_t feature; float value; };
struct Example   { size_t size; float label; Datapoint *dp; };
struct Partition {
    Example    *examples;    // pointer to examples array in 'data'
    Datapoint *datapoints;   // pointer to datapoints array in 'data'
    double     *model;       // pointer to model vector insize 'data'
    size_t      numExamples; // number of examples
    char       *data;        // contains all data (examples, datapoints, model)
    size_t      size;        // total size of memory allocated for 'data'
};
```

Listing 1: In-memory data structures of Chicle

While we have considered an anticipatory background transfer mechanism, our evaluations (see Table 3) show that the overhead, introduced by our mechanism, does not necessitate this.

4 Evaluation

In our evaluation, we attempt to answer the question of how much the CoCoA algorithm can be improved by scaling-in training and thus staying in front of the *knee* for as long as possible.

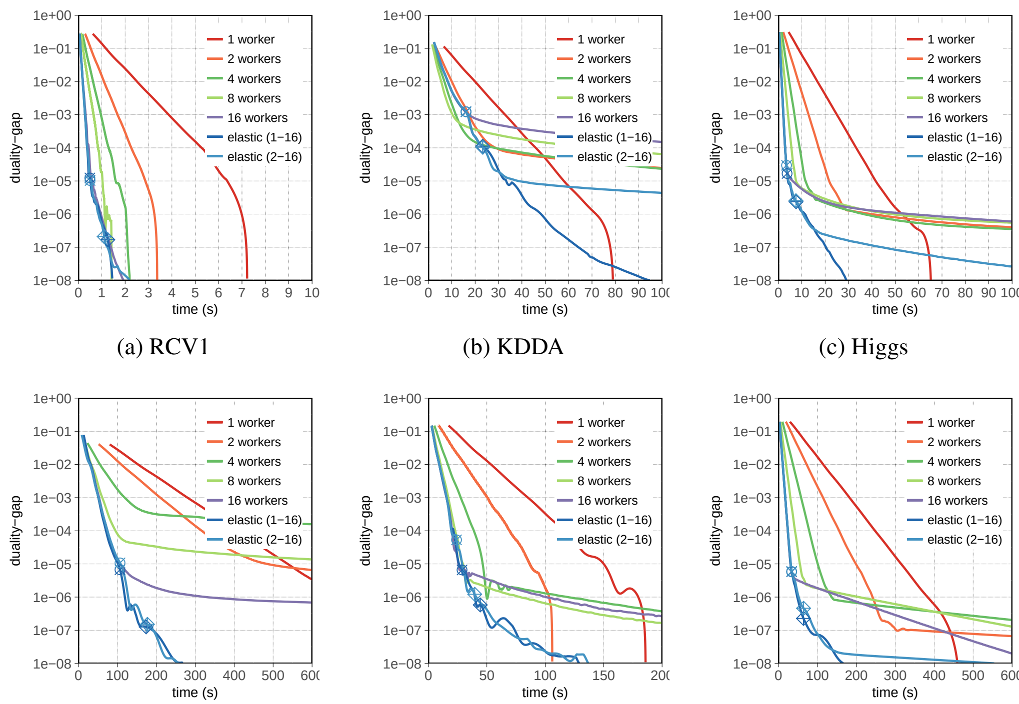
To answer this question, we compare the time-to-accuracy (duality-gap) of our static CoCoA implementation with our elastic version, using an SVM training algorithm[4] and the 6 datasets shown in Table 1. We evaluate static settings with 1, 2, 4, 8 and 16 workers as well as two elastic settings. In the first elastic setting, we start with 16 workers and scale-in to a single worker. This represents cases where the entire dataset fits inside a single node's memory but limited CPU resources make distribution beneficial anyway. In the second elastic setting, we start with 16 workers but scale-in to only two workers. This represents cases where a dataset exceeds a single node's memory capacity and a model therefore *cannot* be trained with non-distributed methods. As convergence behavior for 2+ nodes is similar (see Figure 3), this also indicates how our method works in a larger cluster, e.g., when scaling from 64 to 8 nodes. All test are executed on a 17 node cluster, equipped with Intel Xeon

[4] we use a constant regularizer term $\lambda = 0.01$

Dataset	Examples	Features	Size (in-memory)	Sparsity
RCV1	667,399	47,236	0.4 GB	0.16 %
KDDA	20,216,830	8,407,752	2.6 GB	1.8e-04 %
Higgs	11,000,000	28	2.5 GB	92.11 %
KDD12	54,686,452	149,639,105	17 GB	2e-05 %
Webspam	350,000	16,609,143	10 GB	0.02 %
Criteo	45,840,617	999,999	15 GB	3.9e-03 %

Table 1: Datasets used in the evaluation

E5-2640v3/E5-2650v2, 160-256 GB RAM and CentOS/Fedora 26 Linux, running 16 workers and 1 driver, connected by a FDR (56 Gb/s) Infiniband fabric. The initial set of nodes is always chosen randomly. The results, shown in Figure 3 and Table 2, represent the best results over 6 test runs for all schemes, to account for potential node speed variations. We set a test time limit of 10 minutes (not including data loading). Time results include computing the duality gap.



(a) RCV1 (b) KDDA (c) Higgs

(d) KDD12 (e) Webspam (f) Criteo

Figure 3: Duality-gap vs. time plots for the evaluated datasets and settings. Circles depict a scale-in from 16 to 4 workers, diamonds a scale-in from 4 to 2 and 1 worker(s), respectively.

Our evaluation shows that the basic concept of Chicle – to adjust the number of workers based on feedback from the training algorithm – has benefits for most evaluated datasets. When scaling down to a single worker, Chicle shows an average speedup of 2× compared to the best static setting and 2.2× when scaling down to two workers. While our method does not improve upon all evaluated settings and target accuracies (e.g., $1e-8$ for KDDA, Webspam, RCV1), the slowdown (compared to the respective best static setting) is tolerable, and speedups are still achieved compared to non-optimal static settings. It is important to note that the optimal static setting is not necessarily known in advance and may require several test runs to determine. Chicle, on the other hand, is able to find an optimal or near optimal setting automatically, which shows its robustness.

Finally, we measured data-copy rates and overhead due to scaling-in. Both metrics include the actual data-transfer, control plane overhead and data deserialization. We measured data transfer rates of up

Dataset	$1e-6$	$1e-7$	$1e-8$
RCV1	1.05 (16)	1.06 (16)	0.98 (8)
KDDA	1.49 (1)	1.12 (1)	0.83 (1)
Higgs	3.21 (4)	3.14 (1)	2.24 (1)
KDD12	2.75 (16)	>3.15	>2.25
Webspam	1.25 (4)	1.43 (2)	0.82 (2)
Criteo	2.82 (4)	3.80 (2)	2.76 (1)

(a) 1-16 workers

Dataset	$1e-6$	$1e-7$	$1e-8$
RCV1	1.31 (16)	1.12 (16)	0.64 (8)
KDDA	>1.28	–	–
Higgs	3.46 (4)	5.96 (16)	>3.63
KDD12	2.57 (16)	>3.12	>2.35
Webspam	1.12 (4)	1.59 (2)	0.77 (2)
Criteo	2.63 (4)	3.23 (2)	>1.08

(b) 2-16 workers

Table 2: Speed-up factor of an elastic vs. the best static setting (the number of workers of the best static setting is given in parentheses) for reaching a target accuracy of $1-e6$ – $1e-8$. In case no static setting has reached the target accuracy within a 10 minute time-limit, we provide a minimum speedup factor and "–" in case neither an elastic, nor a static setting has reached a target accuracy.

Setting	RCV1	KDDA	Higgs	KDD12	Webspam	Criteo
1-16 workers	0.12 s	0.73 s	0.71 s	5.04 s	2.78 s	4.52 s
2-16 workers	0.06 s	0.39 s	0.38 s	2.78 s	1.53 s	2.18 s

Table 3: Total average scale-in overhead

to 5.8 GiB/s (1.4 GiB/s on average) and overheads as shown in Table 3. As the measured times do not constitute a significant overhead on our system, we did not implement background data transfer. For slower networks, such a method could be used to hide data transfer times behind regular computation.

5 Related Work

To our knowledge Chicle is the first elastic CoCoA implementation. Several other elastic ML systems exist, but in contrast to Chicle, they target efficient resource utilization rather than reducing overall execution time. Litz [6] is an elastic ML framework that over-partitions training data into $P = n \times K$ partitions for K physical workers. Elasticity is achieved by increasing or decreasing the amount of partitions per node. In contrast to Chicle, Litz does not scale based on feedback from the training algorithm nor does it improve the per-epoch training algorithm convergence rate when doing so, as partitions are always processed independently of each other. SLAQ [11] is a cluster scheduler for ML applications. Like Chicle, SLAQ uses feedback from ML applications, but instead of optimizing the time to arbitrary accuracy for one application, SLAQ tries minimize the time to low accuracy for many applications at the same time, by shifting resources from applications with low convergence ratse to those with high ones, assuming that resources can be used more effectively there. Proteus [3] enables the execution of ML applications using transient revocable resources, such as EC2's spot instances, by keeping worker state minimal at the cost of increased communication.

6 Conclusion and Future Work

In this paper we have shown experimentally, that the optimal number of workers for CoCoA changes over the course of the training. Based on this observation we built Chicle, an elastic ML framework, and have shown that it can outperform static CoCoA for several datasets and settings by a factor of 2–2.2$\times$ on average, often, while using fewer resources. Future work includes additional ways to dynamically optimize CoCoA in terms of training time and resource usage, as well as related use-cases, e.g. neural networks [5]. Furthermore, we are working towards a theoretical foundation of our observations.

References

[1] DÜNNER, C., FORTE, S., TAKÁČ, M., AND JAGGI, M. Primal-dual rates and certificates. *arXiv preprint arXiv:1602.05205* (2016).

[2] DÜNNER, C., PARNELL, T. P., SARIGIANNIS, D., IOANNOU, N., ANGHEL, A., AND POZIDIS, H. Snap ML: A hierarchical framework for machine learning. *CoRR abs/1803.06333* (2018).

[3] HARLAP, A., TUMANOV, A., CHUNG, A., GANGER, G. R., AND GIBBONS, P. B. Proteus: agile ml elasticity through tiered reliability in dynamic resource markets. In *Proceedings of the Twelfth European Conference on Computer Systems* (2017), ACM, pp. 589–604.

[4] JAGGI, M., SMITH, V., TAKAC, M., TERHORST, J., KRISHNAN, S., HOFMANN, T., AND JORDAN, M. I. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3068–3076.

[5] LIN, T., STICH, S. U., AND JAGGI, M. Don't use large mini-batches, use local sgd. *CoRR abs/1808.07217* (2018).

[6] QIAO, A., AGHAYEV, A., YU, W., CHEN, H., HO, Q., GIBSON, G. A., AND XING, E. P. Litz: An elastic framework for high-performance distributed machine learning.

[7] SIKDAR, S., TEYMOURIAN, K., AND JERMAINE, C. An experimental comparison of complex object implementations for big data systems. In *Proceedings of the 2017 Symposium on Cloud Computing* (New York, NY, USA, 2017), SoCC '17, ACM, pp. 432–444.

[8] SMITH, V., FORTE, S., MA, C., TAKÁČ, M., JORDAN, M. I., AND JAGGI, M. Cocoa: A general framework for communication-efficient distributed optimization. *JMLR 18* (2018), 1–49.

[9] STUEDI, P., TRIVEDI, A., PFEFFERLE, J., STOICA, R., METZLER, B., IOANNOU, N., AND KOLTSIDAS, I. Crail: A high-performance i/o architecture for distributed data processing. *IEEE Data Eng. Bull. 40*, 1 (2017), 38–49.

[10] ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 10–10.

[11] ZHANG, H., STAFMAN, L., OR, A., AND FREEDMAN, M. J. Slaq: quality-driven scheduling for distributed machine learning. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), ACM, pp. 390–404.