Distributed Frank-Wolfe under Pipelined Stale Synchronous Parallelism

Thomas Peel & Nam-Luc Tran

EURA NOVA

{thomas.peel, nam-luc.tran}@euranova.eu

Introduction

Big Data analytics have reached a certain level of maturity in both industry and science. Within the context of Big Data distributed processing frameworks, iterative machine learning algorithms are usually implemented through the Bulk Synchronous Parallel (BSP) paradigm. This is namely the case for Twister, Huloop and Scala-Graph.

From the recent evolutions in data centers towards the unification of computing resources, we are witnessing the rise of resource managers such as Mesos, Yarn, Omega and Cora. These define the concept of data center operating system as illustrated on Figure 1.

Stale Synchronous Parallelism (SSP) for convergent and iterative algorithms

Coming back to iterative-convergent tasks, that means that a BSP worker running on a machine is not isolated anymore from other frameworks in this cluster. It leads to situations where a worker might temporarily be overloaded with other tasks, identified as a straggler [2, 3]. They show that the BSP performance dramatically suffers in the presence of stragglers and propose the Stale Synchronous Parallel programming (SSP) as illustrated on Figure 2. In this paradigm the synchronization barrier is relaxed in each super-step and faster workers continue to iterate on stale versions of the model stored in a parameter server. Convergence and correctness of the algorithm are however still guaranteed for iterative convergent algorithms [2].

Distributed Frank-Wolfe under SSP

The Frank-Wolfe algorithm [4] is a simple yet powerful algorithm targeting the following optimization problem:

$$\min_{\alpha \in \mathbb{R}^n} f(\alpha),$$

where the function $f$ is convex and continuously differentiable, and the domain $\mathcal{D}$ is a compact convex subset of $\mathbb{R}^n$. Algorithm 1 shows the basic Frank-Wolfe algorithm.

Algorithm 1: Frank-Wolfe algorithm

1. Let $x = 0$.
2. while $f(x) - f^* > \varepsilon$ do
3. \quad $\alpha = \mathop{arg\min}_{\alpha \in \mathcal{D}} \{ f(\alpha) + \nabla f(x)^T \alpha \}$
4. \quad $x = x - \alpha$
5. end for
6. return $x$

In [1], the authors propose a distributed version of this algorithm to solve a separable variant of the problem stated in Eq. (1):

$$\min_{\alpha \in \mathbb{R}^n} f(\alpha) s.t. \| \alpha \| \leq \beta,$$

where $f(\alpha) = g(A\alpha)$ for an atom matrix $A = [a_1, \ldots, a_r] \in \mathbb{R}^{m \times n}$. We consider the column-wise partitioning of $A$ across a set of $k$ worker nodes $V = \{v_i\}_{i=1}^k$. A node $v_i$ given a set of columns denoted by $A_i$, such that $A = \bigcup_i A_i$, $A_i \cap A_j = \emptyset \forall i \neq j$. This formulation is tightly related to optimizing over an atomic set as mentioned in [5].

The algorithm proposed in [1] relies on three steps: a first one that computes locally the best atom $a_i$ and broadcast it to its co-workers, a second one that elects the best atom among all those candidates and a third one where each worker updates the parameter $\alpha$ given the best atom at current iteration.

Algorithm 2: Stale Synchronous Distributed Frank-Wolfe Algorithm

1. Let $x = 0$.
2. for all worker nodes $v_i \in V$ do $x^{(v_i)}$ parallel do
3. \quad $\alpha^{(v_i)} = \mathop{arg\min}_{\alpha \in \mathcal{D}} \{ f^{(v_i)}(\alpha) + \nabla f^{(v_i)}(x^{(v_i)})^T \alpha \}$
4. \quad $x^{(v_i)} = x^{(v_i)} - \alpha^{(v_i)}$
5. end for
6. return $x^{(v_i)}$

We consider a setting where stragglers randomly appear among the workers. Thus, an unbalanced partitioning like the one proposed in [1] is not well suited and dynamic load-balancing scheduling policies are costly to obtain. We study the asynchronous setting where each worker can use a locally optimal atom in order to update its current possibly out-of-date, but with bounded staleness, version of the global solution. Our claim is that avoiding the synchronous update step can help the algorithm to be tolerant to the straggler problem without sacrificing convergence rate. Algorithm 2 formalizes the use of the SSP paradigm to reach our goal.

Application : LASSO regression

In this work we focus on the LASSO algorithm [7]. LASSO is a linear regression method for solving the following sparse approximation problem:

$$\min_{\alpha \in \mathbb{R}^n} \frac{1}{2} \| y - A\alpha \|_2^2 s.t. \| \alpha \| \leq \beta,$$

where we seek to approximate the target value $y$, for the training point i by a sparse linear combination of its features $\alpha_i$, using the same small number of features for all data points.

Duality Gap and Line Search

For the LASSO problem, the duality-gap is fast to compute. Given that $\alpha$ is a minimizer of the linearised problem at point $\alpha$, the duality gap is given by $(\alpha - \alpha^*)^T g(\alpha)$, where $\alpha^*$ is an optimal solution. This quantity only depends on information that is available at each worker. Moreover, the solution to the line search problem can be obtained analytically with nearly no additional computational cost. Indeed, the optimal step-size is obtained by solving:

$$\gamma^* = \arg \min_{\gamma \in \mathbb{R}} \{ (\alpha - \alpha^*)^T g(\alpha) \}$$

The optimal step-size involves the duality-gap value and $g(\alpha)$ which can be evaluated efficiently (because $\alpha = \alpha^*$ is available as it is involved in the duality-gap computation).

Preliminary results

The following preliminary results were obtained on a 5-nodes cluster running Apache Flink with our SSP implementation.

Future directions

The results shown here are encouraging. They show that the Distributed Frank-Wolfe algorithm under SSP empirically converges and is faster than its BSP counterpart. This is especially the case when random nodes in the cluster are under charge, which mimics the setup of a data center OS. Our future direction of research includes the theoretical proof for the convergence, the study of the more general context of optimizing over atomic set, the study of the sparsity of the iterates (away steps might be useful) and the comparison of our solution with other asynchronous approaches like [8, 9].

References