The Guide of the ARock Codebase

March 9, 2016

Background

The fixed-point problem

- operator $T : \mathcal{H} \to \mathcal{H}$
- **problem:** find $x \in \mathcal{H}$ such that

$$x = Tx$$

- abstracts many problems:
  - linear equations
  - convex optimization
  - statistical regression
  - optimal control

ARock[1]: Async-parallel coordinate update

- $\mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_m$
- p agents, possibly $p \neq m$
- each agent randomly picks $i \in \{1, \ldots, m\}$ and updates just x_i:

$$
\begin{aligned}
x_1^{k+1} &\leftarrow x_1^k \\
&\vdots \\
x_i^{k+1} &\leftarrow x_i^k - \eta_k S_i \hat{x}^k \\
&\vdots \\
x_m^{k+1} &\leftarrow x_m^k
\end{aligned}
$$

where $S = I - T$.

[1] Peng-Xu-Yan-Y.'15

What's wrong with the old code?

- It is cluttered;
- It is not portable;
- Creating a new app is troublesome.

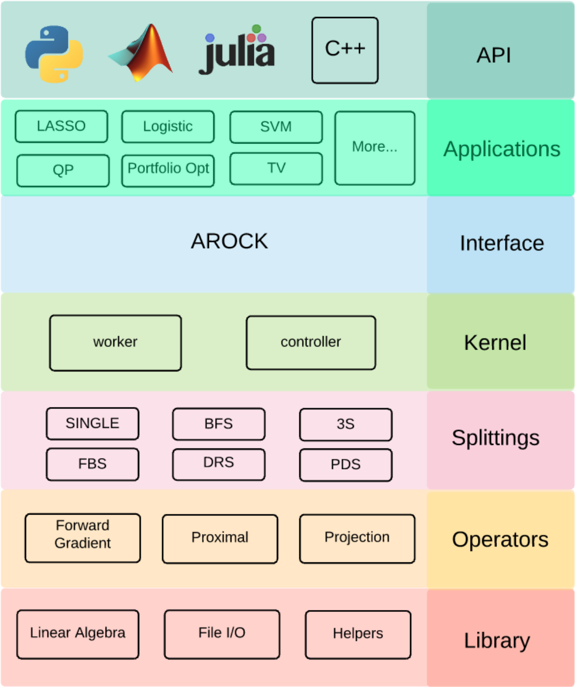
The goal of the new code base

- Be able to create a new app quickly;
- Be able to test new asynchronous ideas easily;
- Be portable and modularized;
- Academic friendly;

Features

- Vector and matrix classes (both dense and sparse);
- A customized and easy-to-use linear algebra library;
- A rich set of coordinate friendly operators (**17** operators);
- Several interfaces for common operator splitting schemes;
- A rich set of applications built on top of ARock;
- A generic interface for controller;
- Matlab, Python and Julia interface (coming soon);
- Runs on Linux, Mac OS X and Windows;
- Approximately 5,000 lines of C++ source code.

Architecture


C++
API
LASSO
Logistic
SVM
More...
QP
Portfolio Opt
TV
Applications
AROCK
Interface
worker
controller
Kernel
SINGLE
BFS
3S
FBS
DRS
PDS
Splittings
Forward Gradient
Proximal
Projection
Operators
Linear Algebra
File I/O
Helpers
Library

Library

Matrix and Vector

- We use the sparse matrix and sparse vector classes from `Eigen`;
- We build our own dense matrix and dense vector classes based on `std::vector`;

Linear Algebra

- Naive implementation is under the `MyAlgebra` namespace;
- Interface for `BLAS` is under the `BLASAlgebra` namespace;

File I/O

- Matrix market format;
- LIBSVM format;
- Matlab format (coming soon)

Helpers

- Objective function;
- Input args parser;

Operator - template

```
struct functor_name {
  // the step_size that associated with the operator
  double step_size;
  // weight on the original function
  double weight;
  // returns the operator evaluated on v at the given index
  double operator() (Vector* v, int index);
  // returns the operator evaluated on val at the given index
  double operator() (double val, int index);
  // full update
  void operator() (Vector* v_in, Vector* v_out);
  // (Optional) update the cached variables
  void update_cache_vars (double old_x_i, double new_x_i,
                                              int index);
  // update the step size
  void update_step_size (double step_size_);
  // customized constructor
  functor_name (double step_size_, double weight_ = 1.);
  // default constructor
  functor_name () : step_size(0.), weight(1.) {}
};
```

Operator - Proximal

$$\mathbf{prox}_f(x) = \arg\min_y f(y) + \frac{1}{2\sigma}\|x-y\|^2$$

Name	Definition	Proximal operator
ℓ_1 norm	$w\|x\|_1$	$shrink(x, w\cdot\sigma)$
sum of squares	$\frac{w}{2}\|x\|_2^2$	$\frac{1}{1+w\cdot\sigma}x$
ℓ_2 norm	$w\|x\|_2$	$\begin{cases} 0 & \text{if } \|x\|_2 \leq w\sigma \\ (1-\frac{w\sigma}{\|x\|_2})\cdot x & \text{otherwise} \end{cases}$
Huber function	$\begin{cases} \frac{w}{2}x^2, & \text{if} -\delta \leq x \leq \delta \\ w\delta(\|x\| - \frac{\delta}{2}), & \text{otherwise} \end{cases}$	$\begin{cases} x - w\sigma\delta & \text{if } x \geq \delta + w\sigma\delta \\ \frac{x}{1+\sigma w} & \text{otherwise} \\ x + w\sigma\delta & \text{if } x \leq -\delta - w\sigma\delta \end{cases}$
elastic net	$w_1\|x\|_1 + \frac{w_2}{2}\|x\|_2^2$	$\frac{1}{1+w_2\sigma}\cdot shrink(x, w_1\sigma)$
log barrier	$-w\sum_i \log(x_i)$	$\frac{1}{2}(x_i + \sqrt{x_i^2 + 4w\sigma}), \forall i$

Operator - Projections

Name	Definition	Projection operator
positive cone	$\{x \mid x \geq 0\}$	$\max(0, x_i), \forall i$
box	$\{x \mid l \leq x \leq u\}, l, u \in R^n$	$\max(l_i, \min(x_i, u_i))$
ℓ_1 ball	$\{x \mid \|x\|_1 \leq r\}$	$O(n \log(n))$ method
ℓ_2 ball	$\{x \mid \|x\|_2 \leq r\}$	$\begin{cases} \frac{r}{\|x\|} \cdot x & \text{if } \|x\|_2 \geq r \\ x & \text{otherwise} \end{cases}$
hyperplane	$\{x \mid a^T x = b\}$	$x + \frac{(b - a^T x)}{a^T a} \cdot a$
probability simplex	$\{x \mid x \geq 0, \sum_i x_i = 1\}$	$O(n \log(n))$ method

Operator - Forward gradient

Name	Definition	Forward gradient operator
square loss	$\frac{w}{2}\|A^Tx - b\|^2$	$x - \sigma w A(A^Tx - b)$
quadratic function	$w(\frac{1}{2}x^TQx + c^Tx + d)$	$x - \sigma w(Qx + c)$
logistic loss	$w\sum_i \log(1 + \exp(-b_i \cdot a_i^Tx))$	$x + \sigma w \sum_i \frac{b_i}{1+\exp(b_i \cdot a_i^Tx)} \cdot a_i$
square hinge loss	$\frac{w}{2}\sum \max(0, b_i(1 - a_i^Tx))^2$	$x + \sigma w \sum b_i \max(0, b_i(1 - a_i^Tx)) \cdot a_i$
square huber loss	$w\sum huber(a_i^Tx - b_i)$	$x - \sigma w \nabla huber(a_i^Tx - b_i)$

Other loss functions:

- error functions in neural networks (coming soon);

Splitting schemes

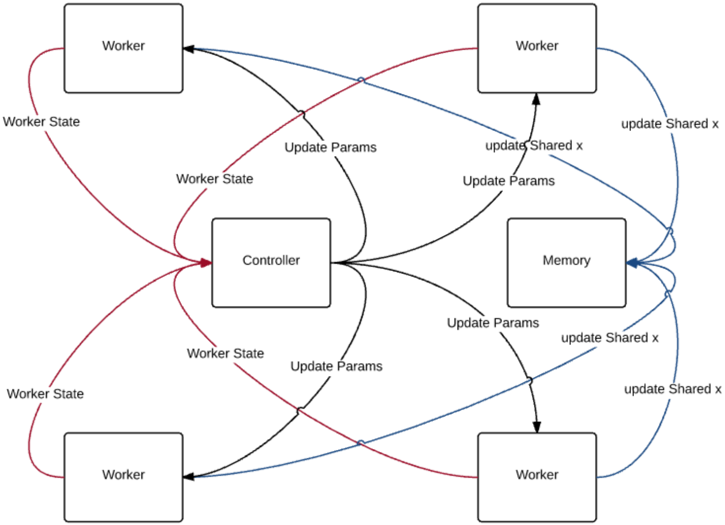
Splitting schemes templated on one or more than one operators. The following are the supported splitting schemes:

- proximal point algorithm;
- forward gradient algorithm;
- forward-backward splitting;
- backward-forward splitting;
- Peaceman-Rachford splitting;
- three operator splitting; (coming soon)
- primal-dual splitting; (coming soon)

Splitting schemes - example

```cpp
template <typename Forward, typename Backward>
struct ForwardBackwardSplitting {
  Forward forward;
  Backward backward;
  Vector* x;
  double relaxation_step_size;
  // constructor
  ForwardBackwardSplitting(Vector*, Forward, Backward);
  // update parameters
  void update_params(Params* params) {
    forward.update_step_size(params->get_step_size());
    backward.update_step_size(params->get_step_size());
    relaxation_step_size = params->get_arock_step_size();
  }
  // make update to x[index], and update the cached variables
  double operator() (int index);
};
```

The two roles: worker and controller



Worker

Worker takes a splitting scheme, and a range of indexes, then runs params->max_itrs of epochs. Other updating orders can be achieved through modifying the worker and the AROCK kernel.

```
template<typename Splitting>
void worker(Splitting algorithm,
            Range range,
            Controller<Splitting>& cont,
            Params* params);
```

Controller

- controller object: defines the interaction between controller and worker.
- controller loop: update FPR and dynamically update the step size.

ARock

Spawn a set of threads to execute the worker function and the controller loop function.

Implemented applications

1. find the intersection of two sets with PRS

$$\min_x \iota_{\{x \mid \|x\|_2 \le 1\}} + \iota_{\{x \mid \|x\|_\infty \le 0.1\}}$$

2. gradient descent for least squares

$$\min_x \frac{1}{2}\|A^T x - b\|^2$$

3. FBS for LASSO

$$\min_x \lambda\|x\|_1 + \frac{1}{2}\|A^T x - b\|^2$$

4. generalized regularized logistic regression

$$\min_x \lambda_1\|x\|_1 + \frac{\lambda_2}{2}\|x\|^2 + \sum_{i=1}^{m} \log(1 + \exp(-b_i \cdot a_i^T x))$$

5. BFS for simple quadratic constrained quadratic programming

$$\min \frac{1}{2}\|A^T x - b\|^2, \text{ s.t. } \|x\| \le 1$$

6. FBS for modified version of SVM (coming soon)

$$\min_x \frac{\lambda}{2}\|x\|^2 + \sum_{i=1}^{m} \max(0, 1 - b_i \cdot a_i^T x)^2$$

Numerical

Platform

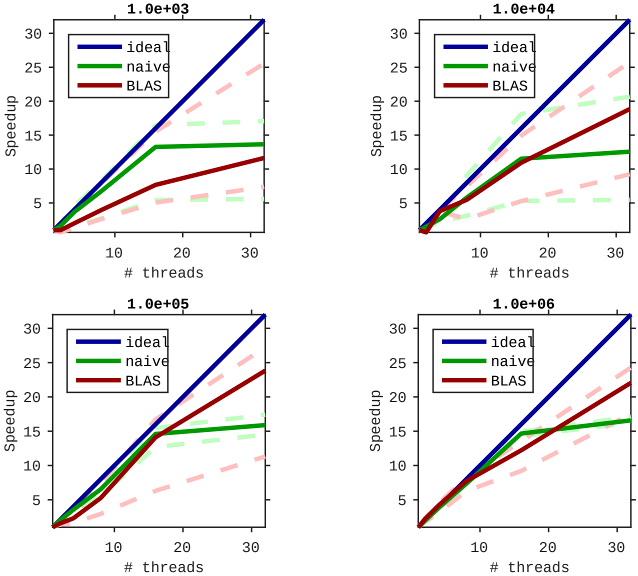
- a node with two Intel Xeon Processors E5-2690 v2 (25M Cache, 3.00 GHz)
- 20 cores, 64 GB of RAM;

Speedup test - baseline tests

- Goal: test the speedup performance of a set of basic operations;
- They will serve as a performance guideline for applying ARock to different applications;
- We implemented the following tests:
  - dot product of two dense vectors;
  - evaluate $\log(1 + \exp(-5.))$;
  - calculate dot$(A(i, :), x)$ for dense A;
  - calculate dot$(A(i, :), x)$ for sparse A;

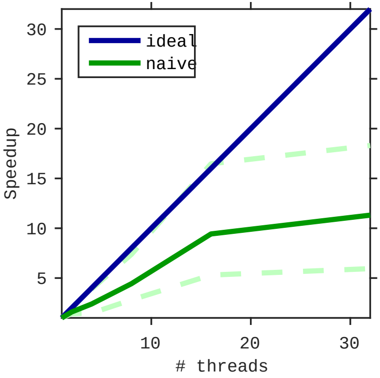
Speedup test - dot product of two dense vectors

- evaluate $x^T y$ for 3200 times.

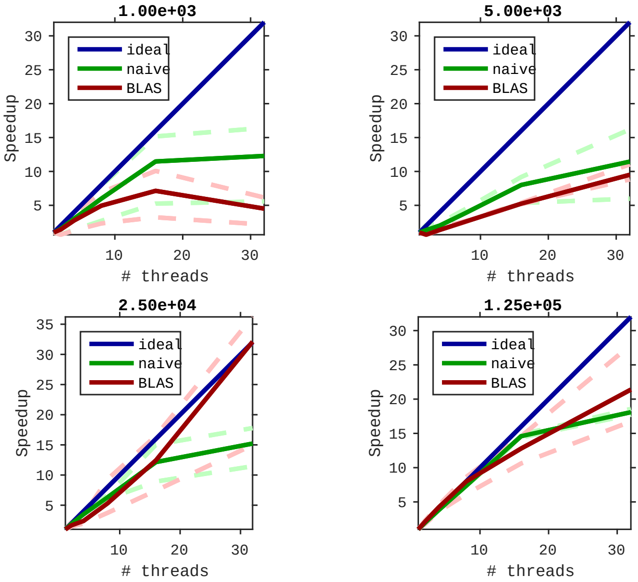


Speedup test - evaluate logistic loss function

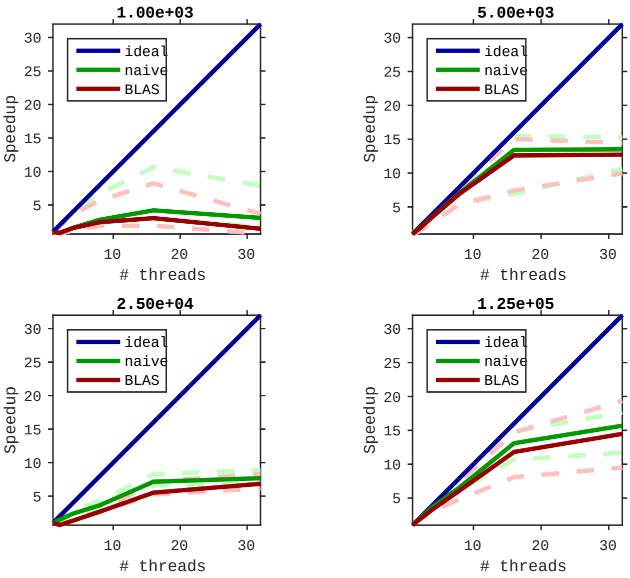
- evaluate $\log(1 + \exp(-8.8))$ for 32 million times.



Speedup test - calculate dot$(A(i,:),x)$ for dense A;



Speedup test - calculate dot$(A(i,:),x)$ for dense A;



Speedup test - sparse logistic regression

# cores	old code		new code	
	Time (s)	Speedup	Time (s)	Speedup
1	27.9	1.0	9.5	1.0
2	17.6	1.6	7.4	1.3
4	9.1	3.1	4.1	2.3
8	5.1	5.5	2.5	3.8
16	3.1	9.0	1.5	6.3
32	2.4	11.6	0.9	10.5

Table : Running times of ARock for the ℓ_1 regularized logistic regression on rcv1.

Future work

- Matlab API;
- Python API;
- more tests with the controller;
- image processing applications with PDS;
- improve the operator implementations;
- more controller schemes.