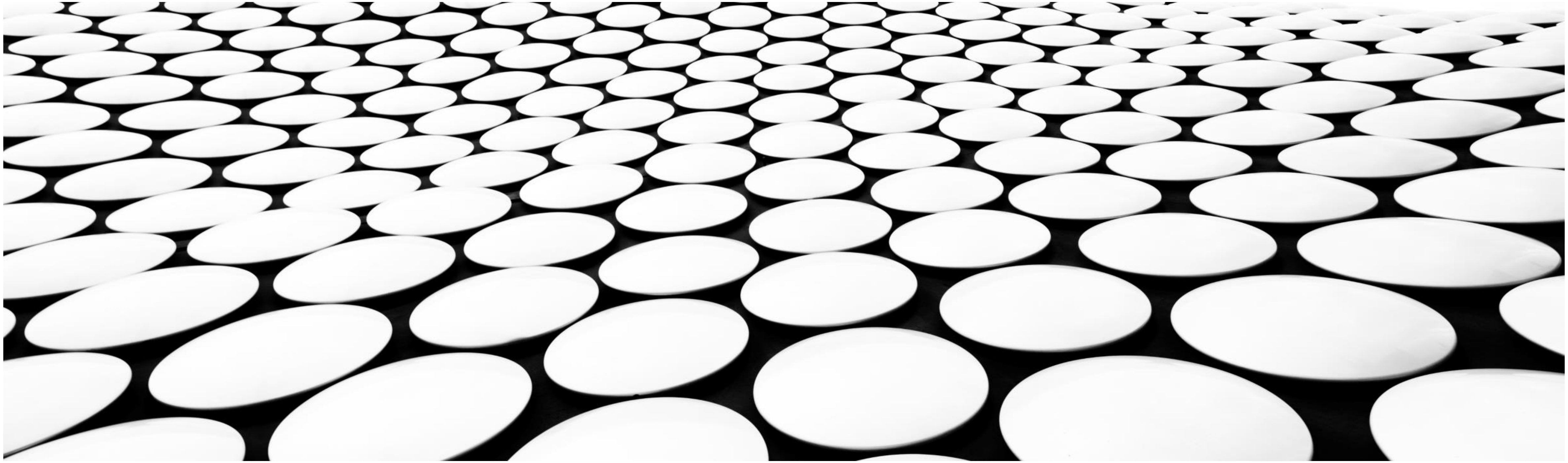
分布式计算

邱怡轩



今天的主题

- 其他分布式算法
- Spark DataFrame

前沿研究

优化算法

- ADMM 提供了一套对模型进行分布式优化计算的框架
- 还有许多新的分布式算法被陆续提出

PPG

- Proximal-Proximal-Gradient 算法
- Ryu, E. K., & Yin, W. (2017). Proximal-proximal-gradient method. arXiv preprint arXiv:1708.06908.

$$\text{minimize} \quad r(x) + \frac{1}{n}\sum_{i=1}^{n}(f_i(x) + g_i(x)),$$

- f_i，g_i，r 为凸函数，f_i 可导

PPG

- 迭代方法

$$
\begin{aligned}
x^{k+1/2} &= \mathbf{prox}_{\alpha r}\left(\frac{1}{n}\sum_{i=1}^{n} z_i^k\right) \\
x_i^{k+1} &= \mathbf{prox}_{\alpha g_i}\left(2x^{k+1/2} - z_i^k - \alpha \nabla f_i(x^{k+1/2})\right) \\
z_i^{k+1} &= z_i^k + x_i^{k+1} - x^{k+1/2},
\end{aligned} \tag{PPG}
$$

- 其中 prox 为近端算子（proximal operator)
- 对于许多常见的函数都有显式解

$$
\mathbf{prox}_h(x_0) = \operatorname*{argmin}_x \left\{ h(x) + \frac{1}{2}\|x - x_0\|_2^2 \right\}
$$

近端分割

- 分布式近端分割算法
- Condat, L., Malinovsky, G., & Richtárik, P. (2020). Distributed proximal splitting algorithms with rates and acceleration. arXiv preprint arXiv:2010.00952.

$$\underset{x\in\mathcal{X}}{\text{minimize}}\ \left\{\Psi(x) := \frac{1}{M}\sum_{m=1}^{M}\Big(F_m(x) + H_m(K_m x)\Big) + R(x)\right\},$$

- F_m ，H_m，R 为凸函数，K_m 为矩阵，F_m 可导

PD3O

Distributed PD3O Algorithm

input: $(\gamma_k)_{k\in\mathbb{N}}$, $\eta \geq \|\widehat{K}\|^2$, $(\omega_m)_{m=1}^M$,
$(q_m^0)_{m=1}^M \in \mathcal{X}^M$, $(u_m^0)_{m=1}^M \in \widehat{\mathcal{U}}$
initialize: $a_m^0 := q_m^0 - K_m^* u_m^0$, $m = 1 \ldots M$
for $k = 0, 1, \ldots$ **do**
  at master, **do**
    $x^{k+1} := \mathrm{prox}_{\gamma_k R}\left(\frac{\gamma_k}{M}\sum_{m=1}^M a_m^k\right)$
    broadcast x^{k+1} to all nodes
  at all nodes, for $m = 1, \ldots, M$, **do**
    $q_m^{k+1} := \frac{M\omega_m}{\gamma_{k+1}} x^{k+1} - \nabla F_m(x^{k+1})$
    $u_m^{k+1} := \mathrm{prox}_{M\omega_m H_m^*/(\gamma_{k+1}\eta)}\Big(u_m^k + \frac{1}{\eta} K_m\big(\frac{M\omega_m}{\gamma_k} x^{k+1} + q_m^{k+1} - q_m^k\big)\Big)$
    $a_m^{k+1} := q_m^{k+1} - K_m^* u_m^{k+1}$
    transmit a_m^{k+1} to master
end for

PDDY

Distributed PDDY Algorithm

input: $(\gamma_k)_{k\in\mathbb{N}}$, $\eta \geq \|\widehat{K}\|^2$, $(\omega_m)_{m=1}^M$, $x_R^0 \in \mathcal{X}$, $(u_m^0)_{m=1}^M \in \widehat{\mathcal{U}}$

initialize: $p_m^0 := K_m^* u_m^0$, $m = 1, ..., M$

for $k = 0, 1, \ldots$ **do**

at all nodes, for $m = 1, \ldots, M$, **do**

$$u_m^{k+1} := \mathrm{prox}_{M\omega_m H_m^*/(\gamma_k \eta)}\big(u_m^k + \frac{M\omega_m}{\gamma_k \eta} K_m x_R^k\big)$$

$$p_m^{k+1} := K_m^* u_m^{k+1}$$

$$x_m^{k+1} := x_R^k - \frac{\gamma_k}{M\omega_m}(p_m^{k+1} - p_m^k)$$

$$a_m^k := M\omega_m x_m^{k+1} - \gamma_{k+1} \nabla F_m(x_m^{k+1}) - \gamma_{k+1} p_m^{k+1}$$

transmit a_m^k to master

at master, **do**

$$x_R^{k+1} := \mathrm{prox}_{\gamma_{k+1} R}\big(\frac{1}{M} \textstyle\sum_{m=1}^M a_m^k\big)$$

broadcast x_R^{k+1} to all nodes

end for

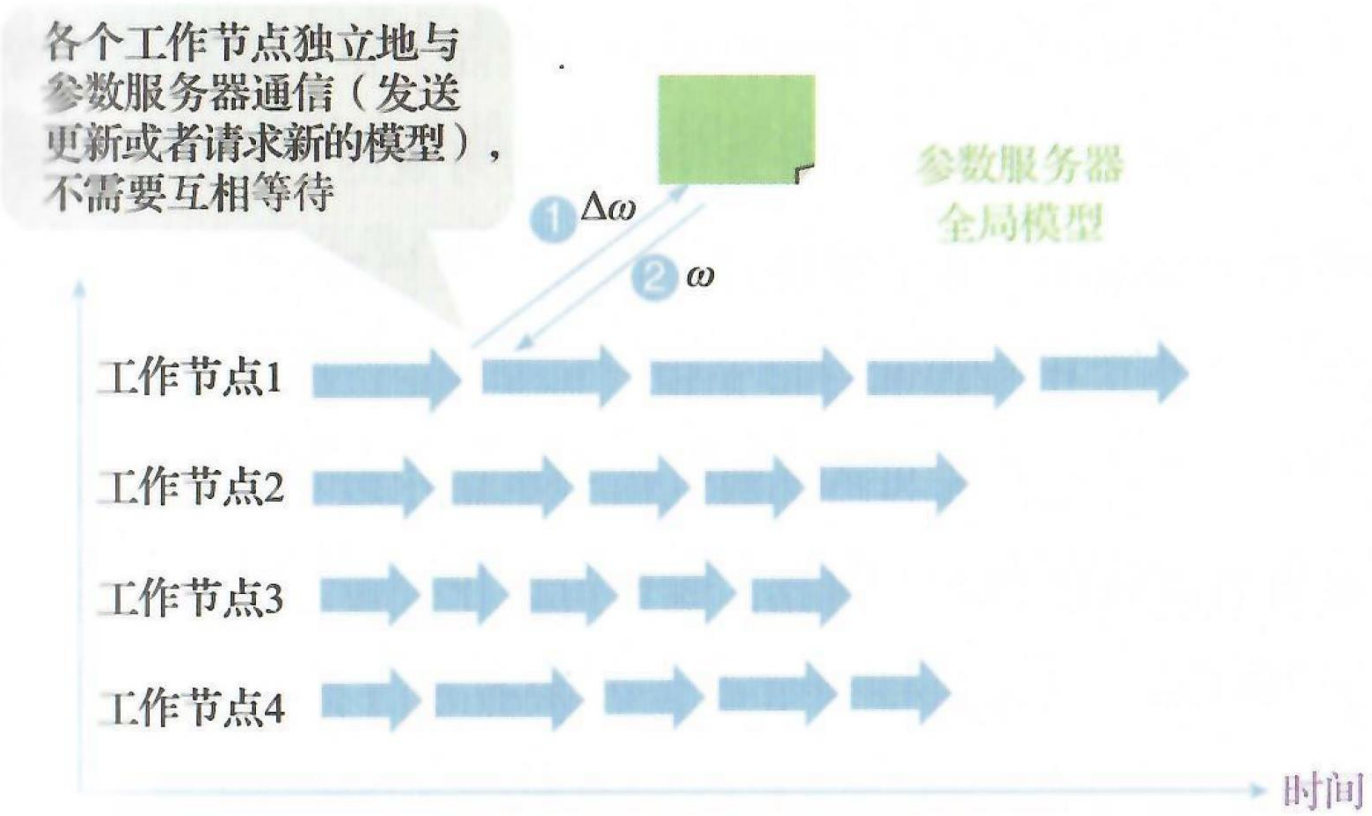
异步算法

- Spark 以及其他 MapReduce 框架适合用来实现同步优化算法
- 即 Map 运算中每个工作节点独立工作
- 而 Reduce 运算要等待所有节点完成

- 好处：保持模型的一致性，有收敛保证
- 坏处：需要等待最慢的“掉队者”

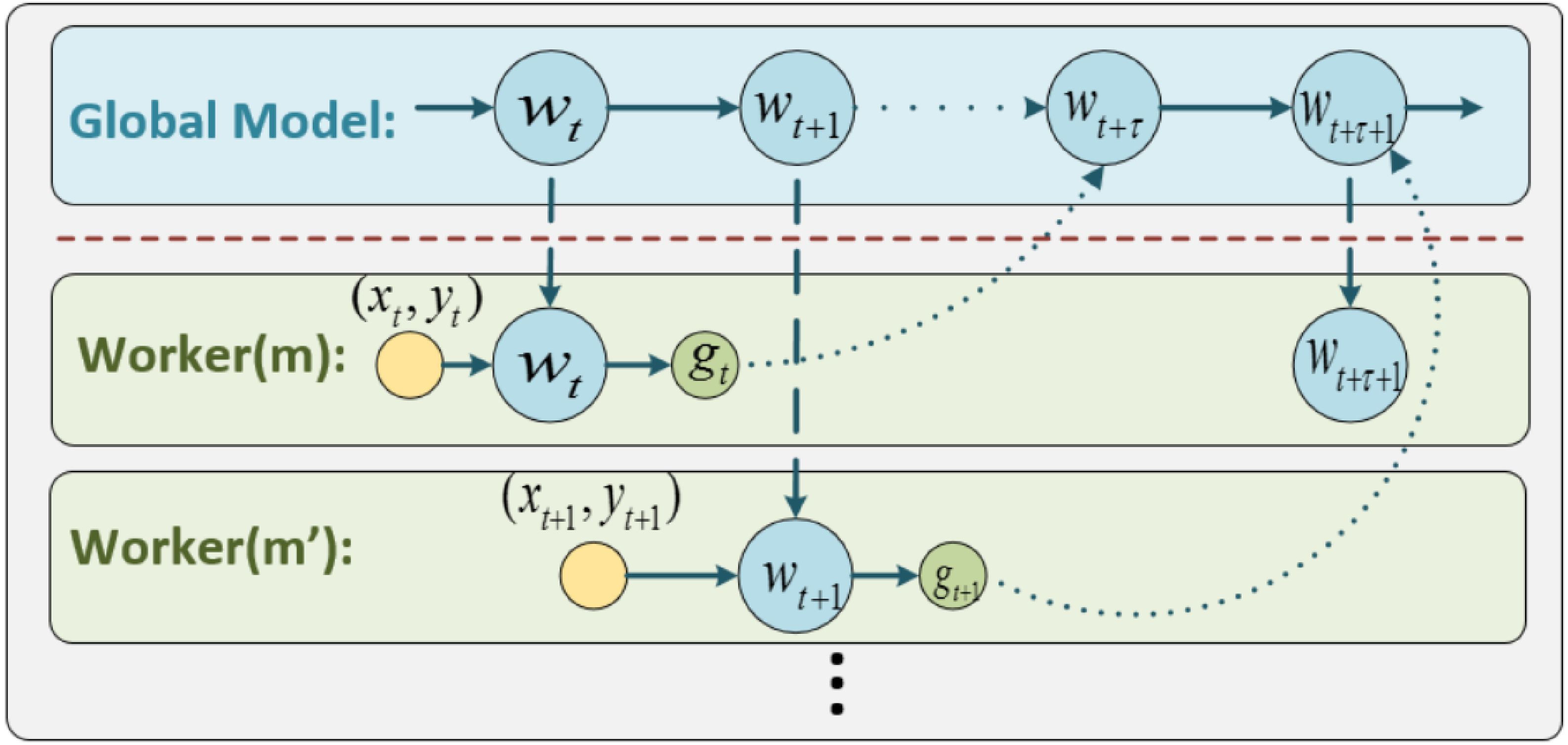
异步算法

- 在异步算法中，每个工作节点可以独立地与主节点通信，无需相互等待



异步算法

- 好处：增大了参数更新的频率
- 坏处：存在“延迟”问题，影响收敛



异步算法

- 异步算法的收敛性质更加复杂，是前沿的研究课题
- 软件框架层面也不如同步算法成熟

Spark DataFrame

DataFrame

- Spark 提供了一种非常有用的数据结构 DataFrame
- 类似于 R 中的 data.frame 和 Python 中的 Pandas
- 但 Spark 中的 DataFrame 是基于 RDD 构建的
- 这意味着它支持分布式的存储和运算

DataFrame

- 参见 lec14-dataframe.ipynb

DataFrame

- 官方文档
- https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html