HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent

Игошин Антон

НИУ ВШЭ

aoigoshin@edu.hse.ru

20 ноября 2017 г.

Overview

Постановка задачи и проблемы

Большой объем данных [100TB] - требуется использовать кластер машин для обработки

- MapReduce - читает данные со скоростью $\sim$ 1 MB/s
- Multicore - читает данные со скоростью $\sim$ 1 GB/s
- Проблема блокировки памяти

Sparse Separable Cost Functions

Наша цель - минимизировать функцию $f : X \subseteq R^n \longrightarrow R$

$$f(x) = \sum_{e \in E} f_e(x_e)$$

где e - маленькое подмножество $\{1, \ldots, n\}$, x_e - значения вектора x на координатах с индексами из e. $|E|$ и n очень большие, но каждая функция f_e работает только на небольшом количестве компонент вектора x.

Такая функция рождает гиперграф $G = (V, E)$, который задает индивидуальные компоненты x. Каждый подвектор x_e содержит $e \in E$ грани графа, содержащий подмножество вершин.

Sparse SVM

Пусть наша задача состоит в обучении SVM для пар данных $E = \{(z_1, y_1), \ldots, (z_{|E|}, y_{|E|})\}$, где $z \in R^n$ и y - ответ для каждой пары $(z, y) \in E$.

$$minimize_x \sum_{\alpha \in E} max(1 - y_\alpha x^T z_\alpha, 0) + \lambda ||x||_2^2$$

Запишем в виде функции стоимости:

$$minimize_x \sum_{\alpha \in E} max(1 - y_\alpha x^T z_\alpha, 0) + \lambda \sum_{u \in e_\alpha} \frac{x_u^2}{d_u}$$

где e_α это ненулевые компоненты в z_α, а d_u - количество объектов с ненулевыми значениями в компонентах u.

Matrix Completion

Данная задача возникает при коллаборативной фильтрации и кластеризации. Здесь Z - низкоранговая матрица с множеством индексов E. Наша задача - перестроить Z из разреженного состояния. Часто используется оценка Z как произведение $LR*$ факторов из следующего:

$$\text{minimize}_{(\boldsymbol{L},\boldsymbol{R})} \sum_{(u,v)\in E} (\boldsymbol{L}_u \boldsymbol{R}_v^* - Z_{uv})^2 + \frac{\mu}{2} \|\boldsymbol{L}\|_F^2 + \frac{\mu}{2} \|\boldsymbol{R}\|_F^2$$

Запишем в виде функции стоимости:

$$\text{minimize}_{(\boldsymbol{L},\boldsymbol{R})} \sum_{(u,v)\in E} \left\{ (\boldsymbol{L}_u \boldsymbol{R}_v^* - Z_{uv})^2 + \frac{\mu}{2|E_{u-}|} \|\boldsymbol{L}_u\|_F^2 + \frac{\mu}{2|E_{-v}|} \|\boldsymbol{R}_v\|_F^2 \right\}$$

где $E_{u-} = \{v : (u, v) \in E\}$ и $E_{-v} = \{u : (u, v) \in E\}$

Graph Cuts

Мы имеем разреженную, неотрицательную матрицу W, которая показывает сходство между компонентами. Наша задача найти такие индексы $\{1, \ldots, n\}$, которые наиболее соответсвуют матрице W. Задача - перевести каждую строку в D-мерное пространство, где каждому объекту соответсвует вектор x_i,

$$S_D = \{\zeta \in R^D : \zeta_v \geq 0 \sum_{v=1}^{D} \zeta_v = 1\}$$

$$minimize_x \sum_{(u,v)\in E} w_{uv} ||x_u - x_v||_1, x_v \in S_D$$

Sparsity notion

$$\Omega := \max_{e \in E} |e|.$$

$$\Delta := \frac{\max_{1 \leq v \leq n} |\{e \in E : v \in e\}|}{|E|}.$$

$$\rho := \frac{\max_{e \in E} |\{\hat{e} \in E : \hat{e} \cap e \neq \emptyset\}|}{|E|}$$

Sparsity notion in examples

Sparse SVM

Δ - частота повторения самого часто встречающегося признака, ρ - мера кластеризуемости гиперграфа

Matrix Completion

$\Delta \approx \frac{log(n_r)}{n_r}$, $\rho \approx \frac{2log(n_r)}{n_r}$

Graph Cuts

Δ is the maximum degree divided by $|E|$, ρ is at most 2Δ

HOGWILD! Algorithm

Algorithm 1 HOGWILD! update for individual processors

loop
  Sample e uniformly at random from E
  Read current state x_e and evaluate $G_e(x_e)$
  for $v \in e$ **do** $x_v \leftarrow x_v - \gamma G_{ev}(x_e)$
end loop

Related work

					HOGWILD!			ROUND ROBIN		
type	data set	size (GB)	ρ	Δ	time (s)	train error	test error	time (s)	train error	test error
SVM	RCV1	0.9	0.44	1.0	9.5	0.297	0.339	61.8	0.297	0.339
MC	Netflix	1.5	2.5e-3	2.3e-3	301.0	0.754	0.928	2569.1	0.754	0.927
	KDD	3.9	3.0e-3	1.8e-3	877.5	19.5	22.6	7139.0	19.5	22.6
	Jumbo	30	2.6e-7	1.4e-7	9453.5	0.031	0.013	**N/A**	**N/A**	**N/A**
Cuts	DBLife	3e-3	8.6e-3	4.3e-3	230.0	10.6	**N/A**	413.5	10.5	**N/A**
	Abdomen	18	9.2e-4	9.2e-4	1181.4	3.99	**N/A**	7467.25	3.99	**N/A**

Experiments

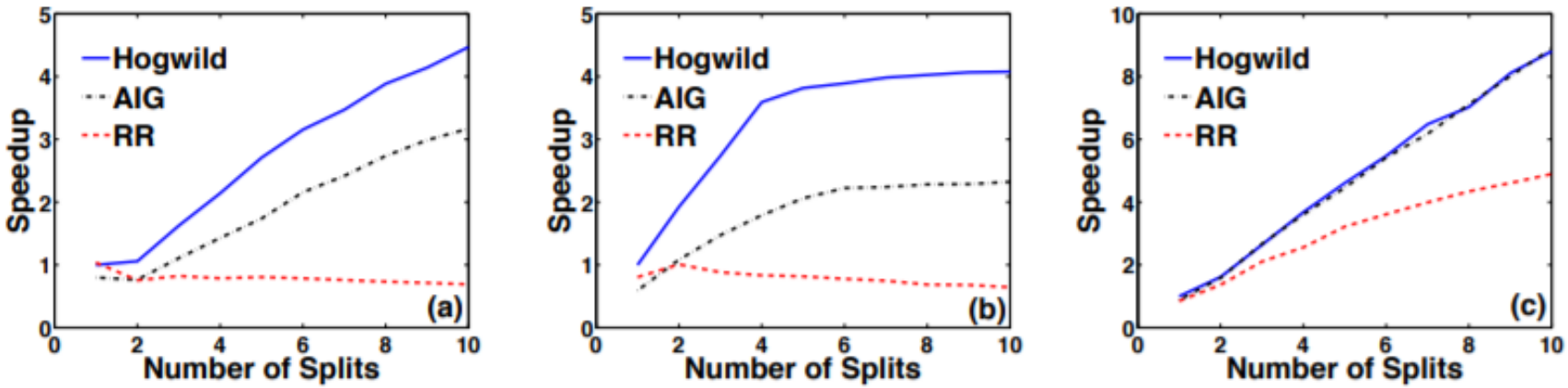

Figure 2: Total CPU time versus number of threads for (a) RCV1, (b) Abdomen, and (c) DBLife.

Experiments

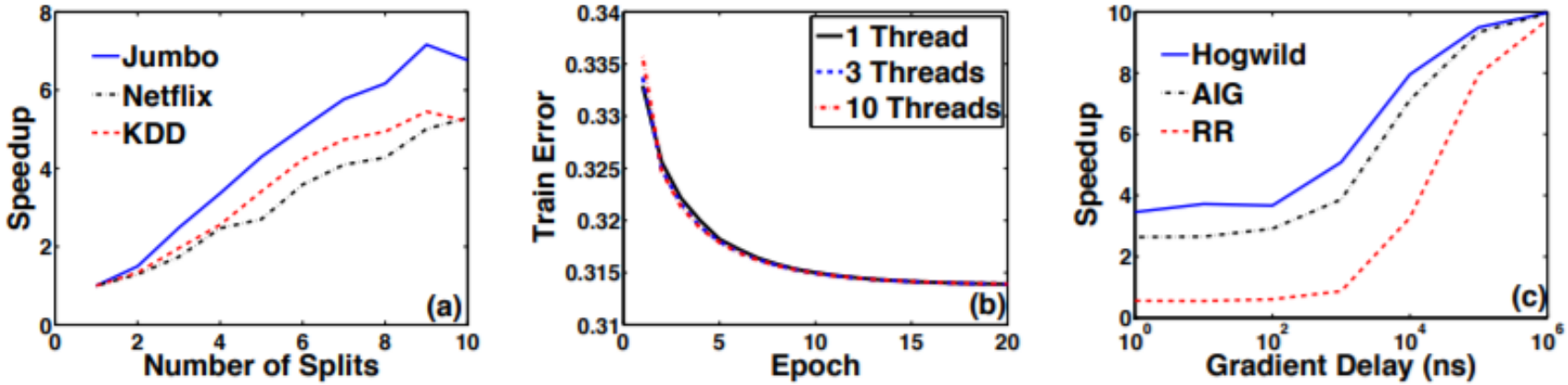

Figure 3: (a) Speedup for the three matrix completion problems with HOGWILD!. In all three cases, massive speedup is achieved via parallelism. (b) The training error at the end of each epoch of SVM training on RCV1 for the averaging algorithm [27]. (c) Speedup achieved over serial method for various levels of delays (measured in nanoseconds).

https://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf , Computer Sciences Department University of Wisconsin-Madison, Madison, WI 53706, June 2011