ECLIPSE: An Extreme-Scale Linear Program Solver for Web-Applications

Kinjal Basu  
LinkedIn AI

Amol Ghoting  
LinkedIn AI

Rahul Mazumder  
MIT

Yao Pan  
LinkedIn AI
Agenda

1. Overview
2. ECLIPSE: Extreme Scale LP Solver
3. Applications
4. System Architecture
5. Experimental Results
Introduction

Large-Scale Linear Programs (LP) has several applications on web
Problems of Extreme Scale

\[ \min_x c^T x \quad \text{s.t.} \quad Ax \leq b \]

- Billions to Trillions of Variables
- Ad-hoc Solutions
  - Splitting the problem to smaller sub-problem \( \rightarrow \) No guarantee of optimality
- Exploit the Structure of the Problem
- Solve a Perturbation of the Primal Problem.
  - Smooth Gradient
  - Efficient computation
Motivating Example

Friend or Connection Matching Problem

- Maximize Value
  - Total invites sent is greater than a threshold
  - Limit on invitations per member to prevent overwhelming members

- $p^1$ - Value Model
- $p^2$ - Invitation Model
- $x_{ij}$ - Probability of showing user $j$ to user $i$

\[
\begin{align*}
\text{Maximize} & \quad \sum_{i,j} x_{ij} p_{ij}^1 \\
\text{subject to} & \quad \sum_{i,j} x_{ij} p_{ij}^2 \geq b_0 \\
& \quad \sum_{i} x_{ij} p_{ij}^2 \leq b_j, \quad j \in \{1, \ldots, J\}, \\
& \quad \sum_{j} x_{ij} = 1, \quad i \in \{1, \ldots, I\}
\end{align*}
\]

Scale:

- $I \approx 10^8$
- $J \approx 10^4$
- $n \approx 10^{12}$

(1 Trillion Decision Variables)
General Framework

\[
\begin{align*}
\min_x & \quad c^T x \\
\text{s.t.} & \quad Ax \leq b \\
& \quad x_i \in C_i, \; i \in [I]
\end{align*}
\]

- Users \( i \), Items \( j \), and \( x_{ij} \) is the association between \((i, j)\)
- \( n = IJ \) can range in 100s of millions to 10s of trillions
- \( C_i \) are simple constraints (i.e. allows for efficient projections)

\[
A = \begin{pmatrix} D_{11} & \cdots & D_{1I} \\ \vdots & \ddots & \vdots \\ D_{m_21} & \cdots & D_{m_2I} \end{pmatrix}
\]

\( A^{(1)} \) Global Constraints
- Cohort Level Constraints
- Eg: Total Invite Constraint

\( A^{(2)} \) Item level constraints
- Eg: Limits on invitation per user
ECLIPSE: Extreme Scale LP Solver
Solving The Problem

Primal LP: \[ P_0^* \ := \ \min_x \ c^T x \quad \text{s.t.} \quad Ax \leq b, \ x_i \in C_i, i \in [I] \]

Old idea: Perturbation of the LP (Mangasarian & Meyer '79; Nesterov '05; Osher et al '11...)

Primal QP: \[ P_\gamma^* \ := \ \min_x \ c^T x + \frac{\gamma}{2} x^T x \quad \text{s.t.} \quad Ax \leq b, \ x_i \in C_i, i \in [I] \]

Dual QP: \[ g_\gamma(\lambda) := \min_{x \in \prod C_i} \left\{ c^T x + \frac{\gamma}{2} x^T x + \lambda^T (Ax - b) \right\} \]

Key Observation: \( \text{length}(\lambda) \) is small

Solve the Dual QP: \[ g_\gamma^* := \max_{\lambda \geq 0} \ g_\gamma(\lambda) = P_\gamma^* \]

Strong duality
Solving The Problem

Primal: \[ P_0^* := \min_x c^T x \quad \text{s.t.} \quad Ax \leq b, \ x_i \in C_i, i \in [I] \]
\[ x^*_\gamma \in \arg\min_x c^T x + \frac{\gamma}{2} x^T x \quad \text{s.t.} \quad Ax \leq b, \ x_i \in C_i, i \in [I] \]

- Observation-1: Exact Regularization (Mangasarian & Meyer ’79; Friedlander Tseng ’08)
  \[ \exists \tilde{\gamma} > 0 \text{ such that } x^*_\gamma \text{ solves LP for all } \gamma \leq \tilde{\gamma} \]

Dual: \[ g_\gamma(\lambda) := \min_{x \in \prod C_i} \left\{ c^T x + \frac{\gamma}{2} x^T x + \lambda^T (Ax - b) \right\} \]
\[ g_\gamma^* := \max_{\lambda \geq 0} g_\gamma(\lambda) \]

- Observation-2: Error Bound (Nesterov ’05)
  \[ |g^*_\gamma - P_0^*| = O(\gamma) \]
Solving The Problem

\[
\max_{\lambda \geq 0} g_\gamma(\lambda)
\]

- Observation-1: Dual objective is smooth (implicitly defined) 
  \[\lambda \mapsto g_\gamma(\lambda) \text{ is } O(1/\gamma)-\text{smooth}\]

- Observation-2: Gradient expression (Danskin’s Theorem)

\[
\nabla g_\gamma(\lambda) = A\hat{x}(\lambda) - b
\]

\[
\hat{x}(\lambda) \in \arg\min_{x \in \Pi C_i} \left\{ c^T x + \frac{\gamma}{2} x^T x + \lambda^T (Ax - b) \right\}
\]

\[
\hat{x}_i(\lambda) = \Pi_{C_i} \left( -\frac{1}{\gamma} (A^T \lambda + c)_i \right)
\]

ECLIPSE Algorithm

- Proximal Gradient Based methods (Acceleration, Restarts)
- Optimal convergence rates.

- Key bottleneck: Matrix-vector multiplication
- Simple projection operation
Overall Algorithm

Input: $A_{m \times n}, \{C_i\}_{i=1}^I, b, c, \gamma$

At Iteration k: Dual $\lambda^k$

Get Primal:

$\hat{x}_i(\lambda^k) = \Pi_{C_i} \left( -\frac{1}{\gamma} (A^T \lambda^k + c)_i \right)$

Compute Gradient:

$\nabla g_\gamma(\lambda^k) = A \hat{x}(\lambda^k) - b$

Update Dual:

GD: $\lambda^{k+1} = (\lambda^k + \eta \nabla g_\gamma(\lambda^k))_+$

AGD: $\lambda^k = (\xi^k + \eta \nabla g_\gamma(\xi^k))_+$

$\xi^{k+1} = \lambda^k + \beta_k (\lambda^k - \lambda^{k-1})$

Next Iteration
Applications
Volume Optimization

Maximize Sessions

- Total number of emails / notifications bounded
- Clicks above a threshold
- Disablement below a threshold

Generalized from global to cohort level systems and member level systems

\[
\max_x \quad x^T p^1 \quad \text{(Total Sessions)} \\
\text{s.t.} \quad x^T 1 \leq c_1 \quad \text{(Sends are Bounded)} \\
\quad x^T p^2 \geq c_2 \quad \text{(Clicks above a threshold)} \\
\quad x^T p^3 \leq c_3 \quad \text{(Disables below a threshold)} \\
\quad 0 \leq x \leq 1 \quad \text{(Probability Constraint)}
\]
Multi-Objective Optimization

- Maximize Metric 1
  - Metric 2 is greater than a minimum
  - Metric 3 is bounded
  - ...

- Most Product Applications

- Engagement vs Revenue
- Sessions vs Notification / Email Volume
- Member Value vs Annoyance

\[
\begin{align*}
\max_x & \quad \sum_{i,j} x_{ij} p_{ij}^1 & \quad \text{(Metric 1)} \\
\text{s.t.} & \quad \sum_{i,j} x_{ij} p_{ij}^2 \geq b_0 & \quad \text{(Metric 2)} \\
& \quad \sum_{i,j} x_{ij} p_{ij}^3 \leq b_1 & \quad \text{(Metric 3)} \\
& \quad \vdots \\
& \quad x_i \in C_i, \ i \in [I]
\end{align*}
\]
System Infrastructure
System Architecture

- Data is collected from different sources and restructured to form Input $A, b, c$
System Architecture

- Data is collected from different sources and restructured to form Input $A, b, c$
- The solver is called which runs the overall iterations.
  - The data is split into multiple executors and they perform matrix vector multiplications in parallel
  - The driver collects the dual and broadcasts it back to continue the iterations
System Architecture

- Data is collected from different sources and restructured to form Input $A, b, c$

- The solver is called which runs the overall iterations.
  - The data is split into multiple executors and they perform matrix vector multiplications in parallel
  - The driver collects the dual and broadcasts it back to continue the iterations

- On convergence the final duals are returned which are used in online serving
### Detailed Spark Implementation

#### Data Representation
- Customized DistributedMatrix API
- $A^{(1)}$: BlockMatrix API from Apache MLLib
- $A^{(2)}$: Leverage Diagonal structure and implement DistributedVector API using RDD (index, Vector)

#### Estimating Primal
- Component wise Matrix Multiplications and Projections are done in parallel
- We cache $A$ in executor and broadcast duals to minimize communication cost.
- The overall complexity to get the primal is $O(J)$

#### Estimating Gradient
- Most computationally expensive step to get $A\hat{x}(\lambda)$
- The worst-case complexity is $O(n = J)$
Experimental Results
Comparative Results

- We compare with a technique of splitting the problem (SOTA):

\[
\begin{align*}
\min_x & \quad c_k^T x \\
\text{s.t.} & \quad A_k x \leq b_k, \quad x_i \in C_i, i \in S_k.
\end{align*}
\]

\[
A = [A_1 : \ldots, A_K]
\]

\[
b = \sum_{k=1}^{K} b_k
\]

\[
c = (c_1, \ldots, c_K)
\]

\[
\hat{\lambda} = \frac{1}{K} \sum_{k=1}^{K} \hat{\lambda}_k
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th>Method</th>
<th>Objective</th>
<th>Primal Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^6$</td>
<td>ECLIPSE</td>
<td>$3.751 \times 10^5$</td>
<td>$6.91 \times 10^{-4}$</td>
</tr>
<tr>
<td></td>
<td>Average 1</td>
<td>$3.748 \times 10^5$</td>
<td>$3.73 \times 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>Average 2</td>
<td>$3.747 \times 10^5$</td>
<td>$1.03 \times 10^{-2}$</td>
</tr>
<tr>
<td>$10^7$</td>
<td>ECLIPSE</td>
<td>$3.750 \times 10^6$</td>
<td>$7.12 \times 10^{-4}$</td>
</tr>
<tr>
<td></td>
<td>Average 1</td>
<td>$3.747 \times 10^6$</td>
<td>$1.71 \times 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>Average 2</td>
<td>$3.747 \times 10^6$</td>
<td>$3.73 \times 10^{-3}$</td>
</tr>
<tr>
<td>$10^8$</td>
<td>ECLIPSE</td>
<td>$3.750 \times 10^7$</td>
<td>$6.56 \times 10^{-4}$</td>
</tr>
<tr>
<td></td>
<td>Average 1</td>
<td>$3.747 \times 10^7$</td>
<td>$1.17 \times 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>Average 2</td>
<td>$3.747 \times 10^7$</td>
<td>$1.73 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

*Table 1. Comparison of our algorithm with the averaging method. Average 1 and 2 correspond to a split size of $10^6$ and $10^7$ respectively.*

Please see the full paper for other comparisons.
Real Data Results

- Test on large-scale volume optimization and matching problems
- Spark 2.3 with up to 800 executors
- 1 Trillion use case converged within 12 hours

<table>
<thead>
<tr>
<th>Problem</th>
<th>Scale $n$</th>
<th>Time (Hours)</th>
<th>ECLIPSE</th>
<th>SCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume Optimization</td>
<td>$10^7$</td>
<td>0.8</td>
<td></td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>$10^8$</td>
<td>1.3</td>
<td></td>
<td>&gt;24</td>
</tr>
<tr>
<td></td>
<td>$10^9$</td>
<td>4.0</td>
<td></td>
<td>&gt;24</td>
</tr>
<tr>
<td>Matching Problem</td>
<td>$10^{10}$</td>
<td>4.5</td>
<td></td>
<td>&gt;24</td>
</tr>
<tr>
<td></td>
<td>$10^{11}$</td>
<td>7.2</td>
<td></td>
<td>&gt;24</td>
</tr>
<tr>
<td></td>
<td>$10^{12}$</td>
<td>11.9</td>
<td></td>
<td>&gt;24</td>
</tr>
</tbody>
</table>

*Table 2. Running time for Extreme-Scale Problems on real data*

Key Takeaways
Key Takeaways

- A framework for solving structured LP problems arising in several applications from internet industry
- Most multi-objective optimization can be framed through this.
- Given the computation resources, we can scale to extremely large problems.
- We can easily scale up to 1 Trillion variables on real data.
Thank you