Resource Elasticity for Large-Scale Machine Learning

Botong Huang[2], Matthias Boehm[1], Yuanyuan Tian[1], Berthold Reinwald[1], Shirish Tatikonda[1], Frederick R. Reiss[1]

[1] IBM Research – Almaden; San Jose, CA, USA
[2] Duke University; Durham, NC, USA

Contact: Matthias Boehm
mboehm@us.ibm.com

SIGMOD 2015
Session #2

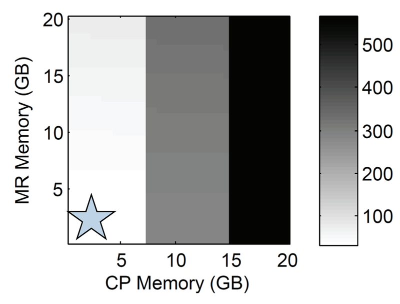
Motivation

Problem Description

- **Declarative Machine Learning**
  - **Goal:** Write ML algorithms independent of input data / cluster characteristics
  - Full flexibility ➔ ML DSL
  - Data independence ➔ Coarse-grained ops
  - Efficiency and scalability ➔ Hybrid plans
  - Specified algorithm ➔ Opt performance
- **Problem of Memory-Sensitive Plans**
  - State-of-the-art compilers sensitive to memory constraints of static cluster configuration
  - **Issue #1:** User needs to reason about plans
  - **Issue #2:** Finding a good cluster configuration is hard (algorithm variety/multi-tenancy)

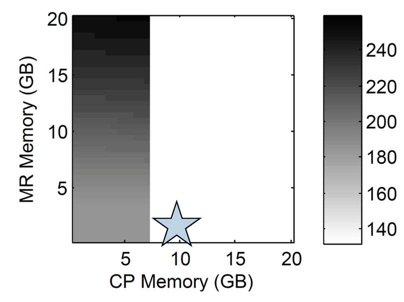
Example: Linear Regression

Linreg DS (Direct Solve)



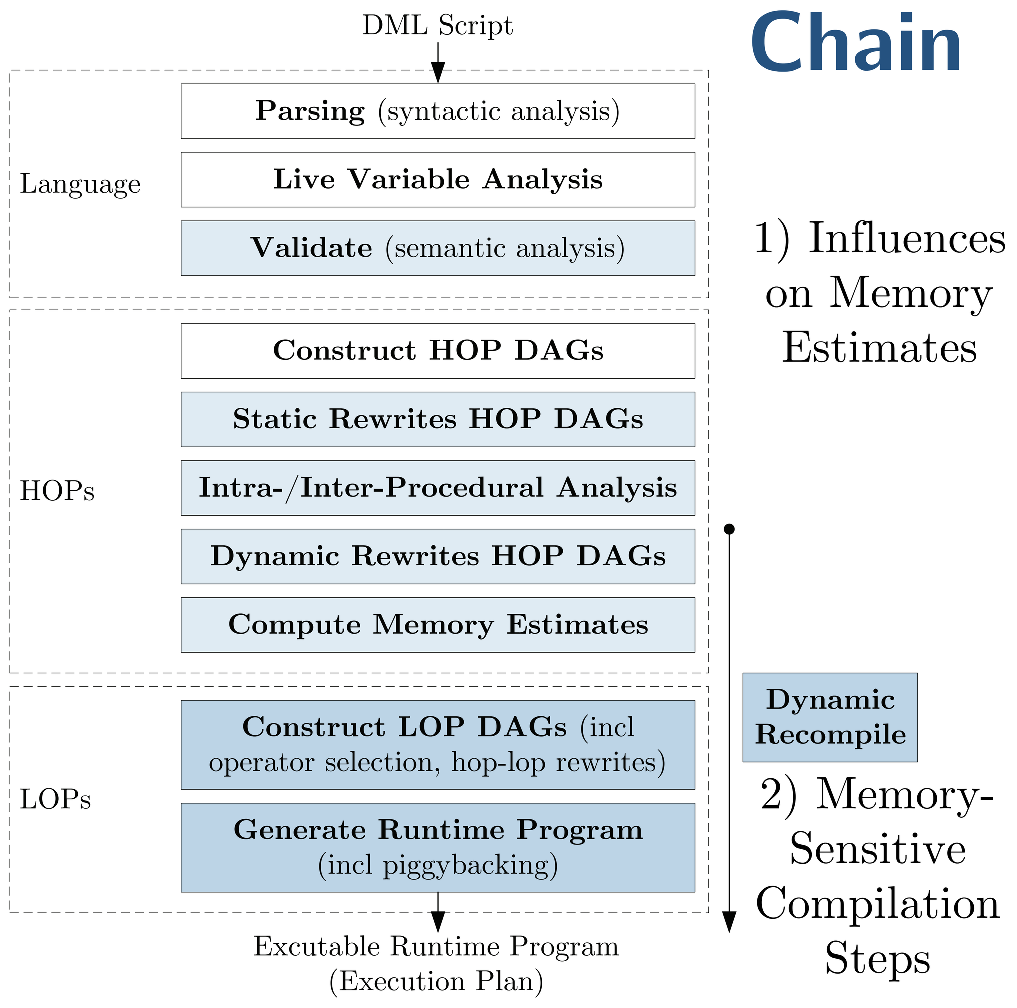
```
X = read($1);
y = read($2);
intercept = $3;
lambda = 0.001;
...
if( intercept == 1 ) {
   ones = matrix(1,nrow(X),1);
   X = append(X, ones);
}
I = matrix(1, ncol(X), 1);
A = t(X) %*% X + diag(I)*lambda;
b = t(X) %*% y;
beta = solve(A, b);
...
write(beta, $4);
```

Linreg CG (Conjugate Gradient)



```
X = read($1);
y = read($2);
intercept = $3;
lambda = 0.001;
...
r = -(t(X) %*% y);
while( i<maxi & norm_r2>norm_r2_trgt ) {
   q = t(X) %*% (X %*% p) + lambda*p;
   alpha = norm_r2 / (t(p) %*% q);
   w = w + alpha * p;
   old_norm_r2 = norm_r2;
   r = r + alpha * q;
   norm_r2 = sum(r * r);
   beta = norm_r2 / old_norm_r2;
   p = -r + beta * p;
   i = i + 1;
}
...
write(w, $4);
```

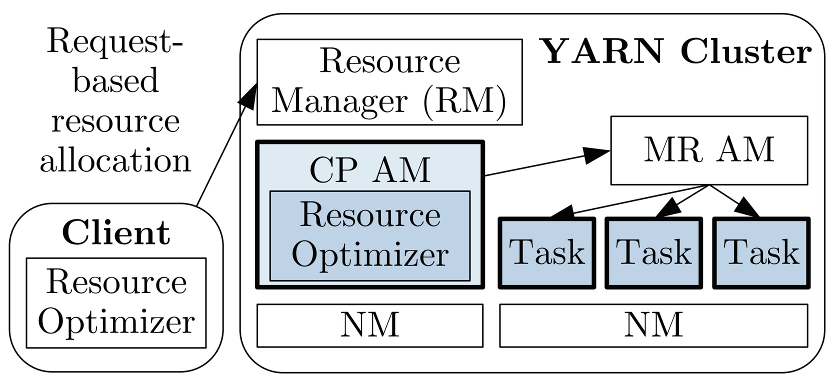
SystemML's Compilation Chain



Resource Optimization

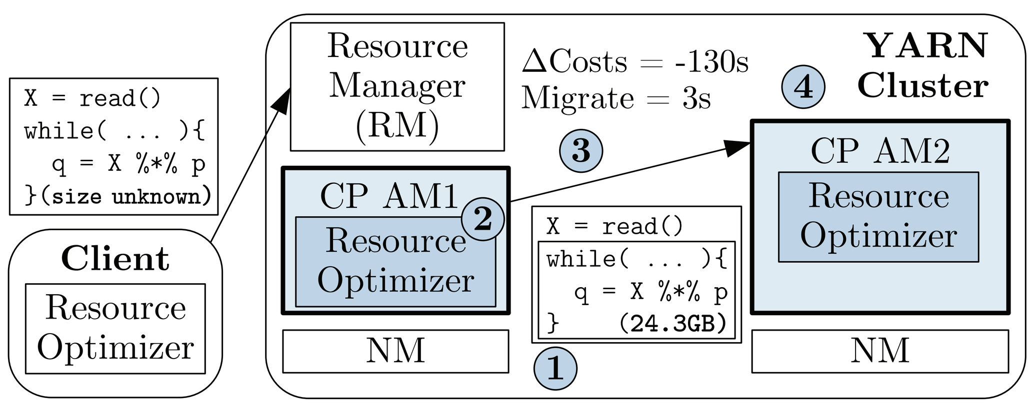
System Architecture

- **SystemML's YARN Integration**
  - Control program (CP) runs as an AM



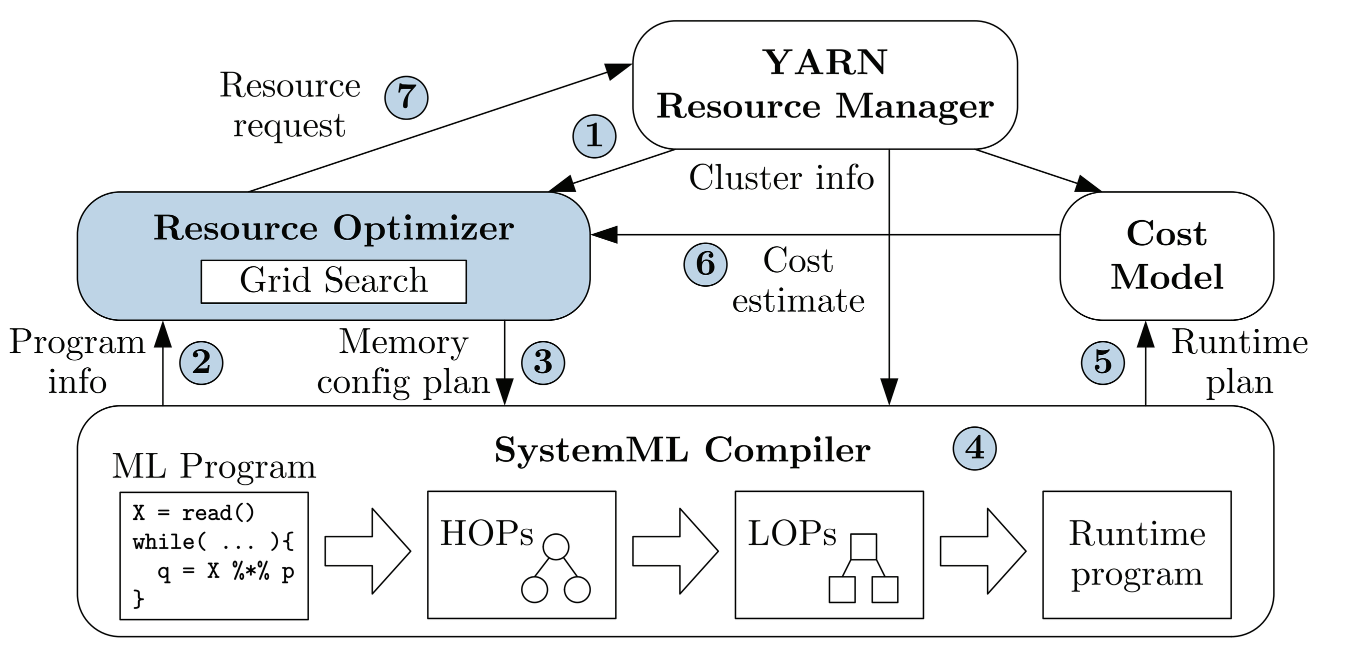
Resource Adaptation

- **Integrated w/ Dynamic Recompilation**
  - (1) Determine re-optimization scope
  - (2) Resource re-optimization
  - (3) Adaptation decision
  - (4) Runtime migration



Resource Optimizer Overview

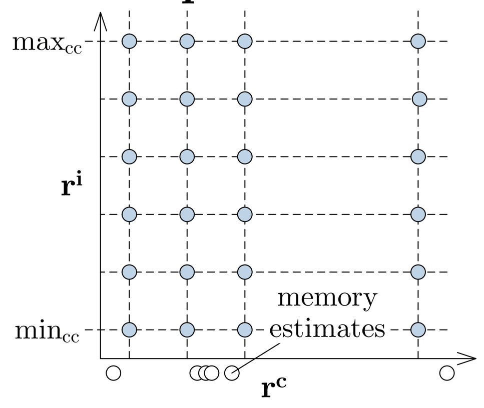
- **Basic Ideas**
  - Optimize ML program resources via online what-if analysis
  - Generating and costing runtime plans
  - Program-aware grid enumeration, pruning, and re-optimization



Grid Enumeration Algorithm

- **Overall Algorithm**
  - Grid enumeration of CP/MR resources
  - Program-aware grid point generation and pruning
- **Grid Point Generators**
  - Systematic: Equi-/Exp-spaced grid
  - Directed: Memory-based grid
  - Composite grids
  - Default: **(Exp ∪ Mem) × (Exp ∪ Mem)**

Equi - Mem



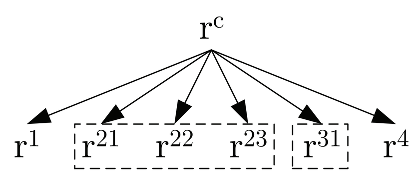
Optimization Framework

- **ML Program Resource Allocation Problem**
  - Goal: Minimize cots w/o unnecessary over-provisioning

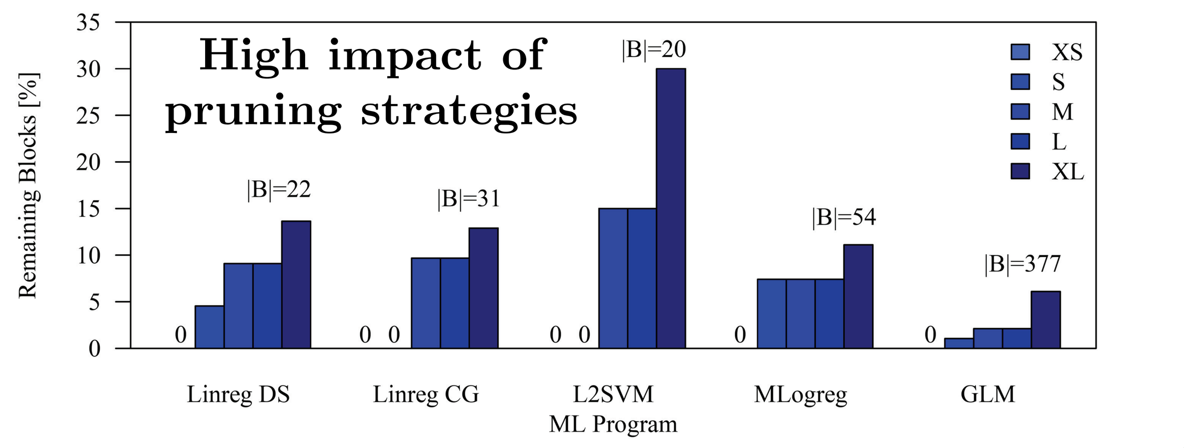
$$\mathcal{R}_P^* = \min\left(\underset{\mathcal{R}_P \in [min_{cc}, max_{cc}]}{\arg\min} C(P, \mathcal{R}_P, cc)\right)$$

➔ **(1) Performance**
➔ **(2) Throughput/Monetary costs**

- **Cost Model**
  - White-box cost model for generated runtime plans
  - C(P, R_P, cc) as estimated execution time of plan P
  - Single pass w/ status tracking of live variables
  - Analytical model of IO, compute (FLOPs), and latency
- **Search Space Characterization**
  - Def: resource dependency (cost influences of resources)
  - Resource configuration $R_P = (r^c_P, r^1_P, ..., r^n_P)$
  - **P1:** Monotonic Dependency Elimination ➔ **Pruning**
  - **P2:** Semi-Indepen. 2-Dim Problems ➔ **Parallelization**



- **Pruning Techniques**
  - Pruning blocks of (1) small ops / (2) unknowns



Experiments (as of 10/2014)

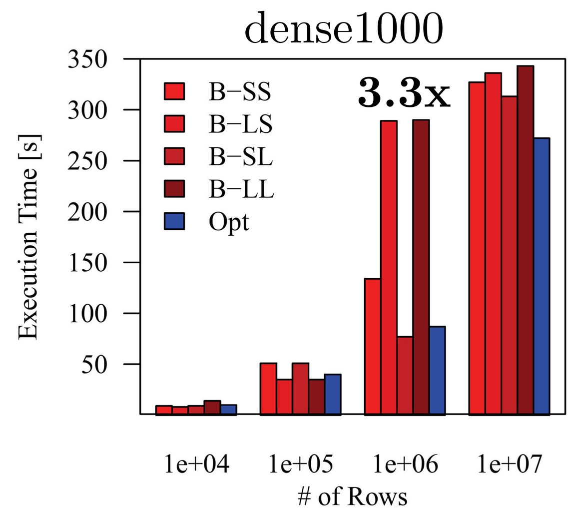
Experimental Setting

- **HW Cluster: 1+6 nodes**
  - 1 head node: 2x4 Intel E5530, HT, 64 GB RAM,
  - 6 worker nodes: 2x6 Intel E5-2440, HT, 96 GB RAM, 12x2 TB disks, 10 Gb Ethernet, Red Hat Enterprise Linux Server 6.5
- **Hadoop Cluster**
  - IBM Hadoop 2.2.0, IBM JDK 1.6.0 64bit SR12
  - YARN max allocation per node: 80 GB (1.5x -Xmx), HDFS block size 128 MB,
- **ML Programs, Data and Baselines**
  - 5 full-fledged ML scripts
  - Dense/sparse (1.0/0.01), 1000/100 features
  - XS-XL (10^7-10^{11} cells), in dense: 80MB – 800GB
  - **B-SS** (512MB), **B-LS** (54GB/512MB), **B-SL** (512MB/54GB), **B-LL** (54GB/4.4GB)

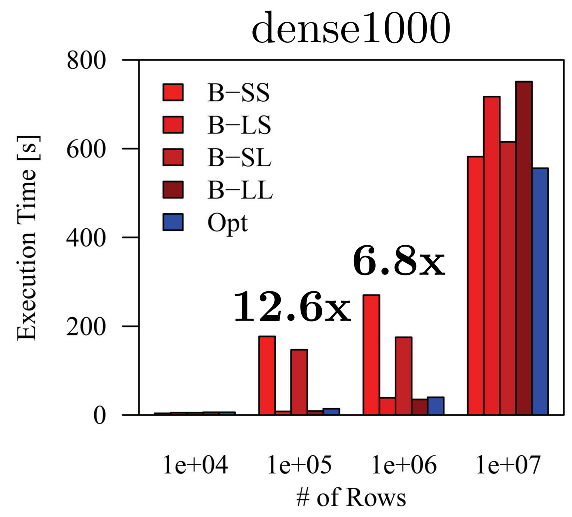
03/2015: IBM BigInsights 4.0 GA, BigR/SystemML (opt level)

End-to-End Runtime

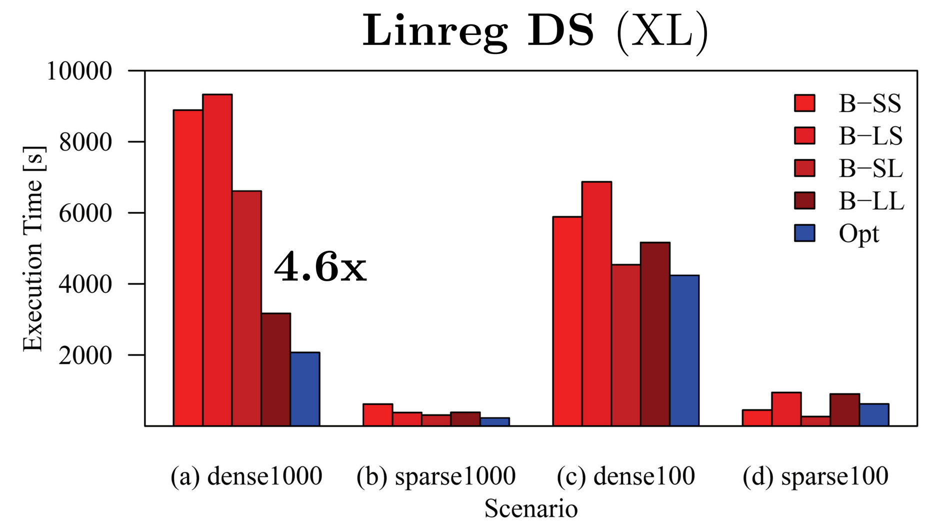
Linreg DS (XS-L), dense1000



Linreg CG (XS-L), dense1000

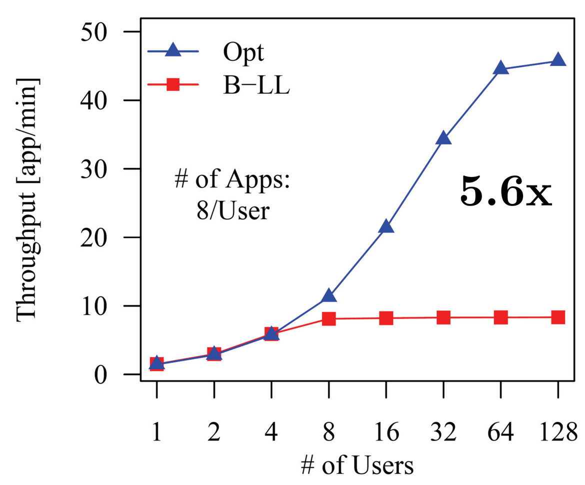


Linreg DS (XL)



End-to-End Throughput

- **Linreg DS** (Scenario S, dense1000)
  - B-LL : 53.3 GB/4.4 GB ➔ max parallelism: 6
  - Opt: 8GB/2GB ➔ 36



Optimization Overhead

- **Linreg DS** (dense 1000)

Scenario	# Comp.	# Cost.	Opt. Time	%
XS (80MB)	352	8	**0.35s**	3.5
S (800MB)	417	30	**0.41s**	1.0
M (8GB)	678	168	**0.71s**	0.8
L (80GB)	448	104	**0.56s**	0.2
XL (800GB)	536	149	**0.73s**	<0.1