Big Data Lab, Colorado State University

DRAW: A New Data – gRouping – AWare Data Placement Scheme for Data Intensive Applications with Interest Locality

Khushboo Gupta

OVERVIEW

- Introduction
- Design of DRAW
- Results and Analysis
- Conclusions

INTRODUCTION

- Without taking data grouping into consideration, the random placement does not perform well and is way below the efficiency of optimal data distribution.
- DRAW extracts optimal data groupings and re-organizes data layouts to achieve the maximum parallelism per group subjective to load balance.

DESIGN OF DRAW

- design DRAW at rack-level, which optimizes the grouping-data distribution inside a rack.
- There are three parts:

1. A data access history graph(HDAG) to exploit system log files learning the data grouping information.
2. A data grouping matrix (DGM) to quantify the grouping weights among the data and generate the optimized data groupings.
3. An optimal data placement algorithm (ODPA) to form the optimal data placement.

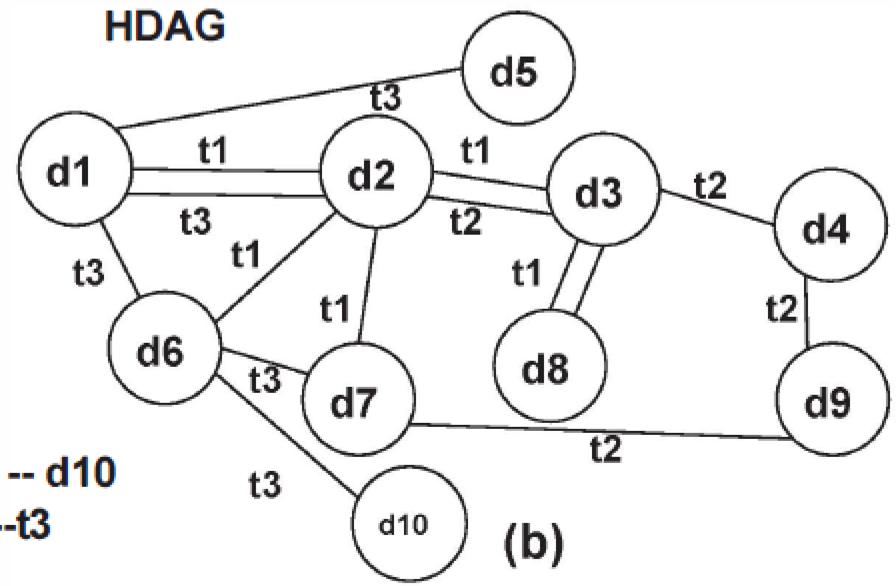
DESIGN OF DRAW

- A. History Data Access Graph (HDAG)

Tasks	Data of interest
t1	{d1,d2,d3,d6,d7,d8}
t2	{d2,d3,d4,d7,d9}
t3	{d1,d2,d5,d6,d7,d10}

(a)

Data: d1 -- d10
Task: t1--t3



(b)

DESIGN OF DRAW

- B. Data Grouping Matrix (DGM)

Data Grouping Matrix (DGM)

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
d1	2	2	1	0	1	2	2	1	0	1
d2	2	3	2	1	1	2	3	1	1	1
d3	1	2	2	1	0	1	2	1	1	0
d4	0	1	1	1	0	0	1	0	1	0
d5	1	1	0	0	1	1	1	0	0	1
d6	2	2	1	0	1	2	2	1	0	1
d7	2	3	2	1	1	2	3	1	1	1
d8	1	1	1	0	0	1	1	1	0	0
d9	0	1	1	1	0	0	1	0	1	0
d10	1	1	0	0	1	1	1	0	0	1

DESIGN OF DRAW

- Bond Energy Algorithm (BEA)
- Find the sub-optimal solution in time $O(n^2)$

Group 1

Clustered DGM (CDGM)

	6	7	2	1	3	10	5	9	8	4
6	3	2	2	2	2	1	1	0	1	1
7	2	3	2	2	2	1	1	1	1	1
2	2	2	3	2	1	1	1	0	1	0
1	2	2	2	2	1	1	1	0	1	0
3	2	2	1	1	2	0	0	1	1	1
10	1	1	1	1	0	1	1	0	0	0
5	1	1	1	1	0	1	1	0	0	0
9	0	1	0	0	1	0	0	1	0	1
8	1	1	1	1	1	0	0	0	1	0
4	1	1	0	0	1	0	0	1	0	1

Sub-matrix for ODPA (OSM)

DESIGN OF DRAW

- C. Optimal Data Placement Algorithm (ODPA)

Algorithm 1 ODPA algorithm

Input: The sub-matrix (OSM) as shown in Figure 3: $M[n][n]$; where n is the number of data nodes;
Output: A matrix indicating the optimal data placement: $DP[2][n]$;
Steps:
for each row from $M[n][n]$ **do**
  R = the index of current row;
  Find the minimum value V in this row;
  Put this value and its corresponding column index C into a set ***MinSet***;
  $MinSet = C1, V1, C2, V2, ;$ // there may be more than one minimum value
  if there is only one tuple $(C1, V1)$ in $MinSet$ **then**
    //The data referred by C1 should be placed with the data referred by R on the same node;
    $DP[0][R] = R;$
    $DP[1][R] = C1;$
    Mark column $C1$ is invalid (already assigned);
    Continue;
  end if
  for each column C_i from $MinSet$ **do**
    Calculate $Sum[i] = sum(M[\star][C_i]);$ // all the items in C_i column
  end for
  Choose the largest value from Sum array;
  C = the index of the chosen Sum item;
  $DP[0][R] = R;$
  $DP[1][R] = C;$
  Mark column C is invalid (already assigned);
end for

DESIGN OF DRAW

- C. Optimal Data Placement Algorithm (ODPA)

Without ODPA, the parralelism may be not maximized

node1	*node2*	*node3*	*node4*	*node5*
d6	d7	d1	d2	d3
d4	d9	d5	d10	d8

Tasks	requried data	Involved nodes
t1	**d1,d2,d3,d6,d7,d8**	1,2,3,4,5
t2	**d2,d3,d4,d7,d9**	***1,2,4,5***
t3	**d1,d2,d5,d6,d7,d10**	***1,2,3,4***

Not optimal

(1)

Optimized data layout maximizes the parallelism

node1	*node2*	*node3*	*node4*	*node5*
d6	d7	d1	d2	d3
d9	d8	d4	d10	d5

Tasks	requried data	Involved nodes
t1	**d1,d2,d3,d6,d7,d8**	1,2,3,4,5
t2	**d2,d3,d4,d7,d9**	1,2,3,4,5
t3	**d1,d2,d5,d6,d7,d10**	1,2,3,4,5

Optimal

(2)

RESULTS AND ANALYSIS

- Test bed consists of 40 heterogeneous nodes on a single rack.
- Genome Indexing and Astrophysics Applications
- Performance Improvement of MapReduce Programs

	Total maps	Local maps	Ratio
On DRAW	399	302	76.1%
On Random	399	189	47.1%

RESULTS AND ANALYSIS

- Overhead of DRAW

1. Building HDAG
2. Building and Clustering DGM : 37 seconds to cluster the 640*640 matrix.
3. Data Re-organization :The overall execution times on randomly placed data and DRAW re-organized data are 33min43sec and 20min37sec.

CONCLUSIONS

- DRAW captures runtime data grouping patterns and distributes the grouped data as evenly as possible.
- DRAW can significantly improve the throughput of local map task execution by up to 59.8%, and reduce the execution time of map phase by up to 41.7%. The overall MapReduce job response time is reduced by 36.4%.

QUESTIONS