Lossless Data Compression
Standard Applications and the MapReduce Web Computing Framework

Sergio De Agostino

Computer Science Department

Sapienza University of Rome

Internet as a Distributed System

Modern distributed systems may nowadays consist of hundreds of thousands of computers.

Decentralization is the fundamental approach to reach unprecedented scales, towards millions or even billions of elements using Internet as an extreme distributed system.

Web Computing

Web Computing is a special kind of distributed computing and involves internet-based collaboration of an arbitrary number of remotely located applications.

MapReduce is the framework proposed by Google to realize distributed applications on the Internet, making distributed computing more accesible.

The MapReduce Framework

MapReduce (MR) is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes).

Nodes could be either on the same local network or shared across geographically and administratively more heterogenous distributed systems.

The MR Programming Paradigm

$P = \mu_1\rho_1 \ldots. \mu_R\rho_R$ is *a* sequence where μ_i is a *mapper* and ρ_i is a *reducer* for $1 \leq i \leq R$.

Mapper μ_i receives the input from reducer ρ_{i-1} for $1 < i \leq R$.

For $1 \leq i \leq R$, such input is a multiset U_{i-1} of *(key, value)* pairs (U_0 is given) and each pair $(k, v) \in U_{i-1}$ is mapped by μ_i to a set of new *(key, value)* pairs.

The MR Programming Paradigm (2)

The input to reducer ρ_i is U_i', where U_i' is the union of the sets output by mapper μ_i.

For each key *k*, ρ_i reduces the subset of pairs with *k* as key component to a new set of pairs with key still equal to *k.*

U_i is the union of these new sets and is output by ρ_i as input to μ_{i+1} .

Distributed Implementation

A key is associated with a processor .

Each pair with a given key is processed by the same node.

More keys can be associated with the same node in order to lower the scale of the distributed memory system of nodes involved in the Web computer network.

Input/Output Phases

The sequence $P = \mu_1\rho_1 \ldots. \mu_R\rho_R$ does not include the I/O phases (μ_1 can be seen as a second step of the input phase) .

Extend P to $\mu_0 \mu_1\rho_1 \ldots. \mu_R\rho_R \mu_{R+1}\rho_{R+1}$.

$\mu_0 \mu_1$ is the input phase.

$\mu_{R+1}\rho_{R+1}$ is the output phase.

Input Phase

There is a unique *(key, value)* pair input to μ_0, where the value component is the input data.

μ_0 distributes the data generating U_0 .

μ_1 distributes the data furtherly or is the identity function.

Output Phase

μ_{R+1} maps the multiset U_R to a multiset where all the pairs have the same key k.

ρ_{R+1} reduces such multiset to the unique pair (k, y).

y is the output data.

Complexity Requirements

- R is polylogarithmic in the input size
- sublinear number of nodes
- sub-linear work-space for each node
- mappers running time is polynomial
- reducers running time is polynomial

Karloff, Suri and Vassilvistkii, A Model of Computation for MapReduce, SODA 2010

Stronger Requirements

R is constant (< 10).

Total running time of mappers and reducers times the number of processors must be equal to sequential time.

The MR implementations of Standard Lossless Data Compression satisfy these stronger requirements with R < 3 and guarantee a linear speed-up.

Lossless Data Compression

Standard lossless data compression applications are based on Lempel-Ziv methods (Zip compressors or Unix Compress and Dos Stuffit).

The Zip family is based on sliding window (SW) compression.

Unix Compress and Dos Stuffit implement the Lempel, Ziv and Welch method (LZW compression).

SW Factorization and Compression

SW factorization: string $S = f_1 f_2 \ldots\ f_i \ldots f_k$ where f_i is either the longest substring occurring previously in $f_1 f_2 \ldots\ f_i$ or an alphabet character.

SW compression is obtained by encoding f_i with a pointer to a previous copy.

Bounded memory factorization: f_i is determined by the last w characters (w=32K in the Zip compressor).

LZW Factorization and Compression

Finite alphabet A, dictionary D initially equal to A, S in A^*.

LZW factorization: $S = f_1 f_2 \ldots f_i \ldots f_k$ where f_i is the longest match with the concatenation of a previous factor and the next character.

LZW compression is obtained by encoding f_i with a pointer to a copy of such concatenation previously added to D.

Bounded Memory Factorizations

LZW-FREEZE: f_i is the longest match with one of the first d factors (d=64K).

LZW-RESTART: f_i is determined by the last (i mod d) factors.

To improve LZW-RESTART the dictionary of d factors is frozen and the restart operation happens when the compression ratio starts deteriorating.

Distributed Memory SW Factorization

The SW factorization process is applied independently to different blocks of the input string (no communication cost).

A block of 300kB guarantees robustness (about *ten times* the window length of a Zip compressor).

1000 processors → 300 megabytes

The MapReduce Implementation

Input X; MR sequence $\mu_0\mu_1\rho_1\mu_2\rho_2$; $R=1.$

μ_0 maps $(0, X)$ to $U_0 = \{(1, X_1), \dots (m, X_m)\}$

μ_1 is the identity function ($U_0' = U_0$)

ρ_1 reduces U_0' to $U_1 = \{(1, Y_1), \dots (m, Y_m)\}$ where Y_i is the SW coding of X_i ,$1 \le i \le m$

μ_2 maps U_1 to $U_1' = \{(0, Y_1), \dots (0, Y_m)\}$

ρ_2 reduces U_1' to $(0, Y)$ where Y= $Y_1 \dots Y_m$ (Y is the output).

Distributed Memory LZW Factorization

If LZW factorization is applied in parallel, blocks of 600kB are needed to guarantee robustness (300kB to learn the dictionary and 300kB to compress staticly).

Static compression is extremely scalable.

According to the scale of the network, different approaches can be described.

The MapReduce implementations require two iterations (R=2).

LZW Static Optimal Factorization

The *dictionary* of *d* factors built by the LZW algorithm is *prefix*, that is, every prefix of a factor is in the dictionary.

Since the dictionary is not *suffix*, the greedy factorization of the input string after the dictionary has been *frozen* is not optimal.

Greedy vs Optimal: Example

string: aba^n

dictionary

a, b, ab, ba^k for $0 < k < n+1$

greedy factorization

$ab, a, a, \ldots.., a$

optimal factorization

a, ba^n

Semi-Greedy (Optimal) Factorization

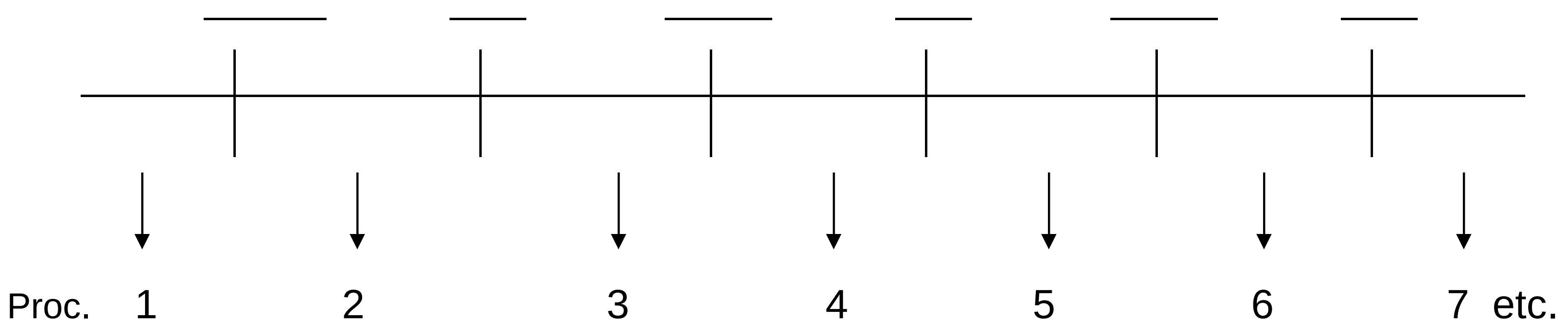
$x_{t+1}, x_{t+2}, \ldots$ is the suffix of the input string after the dictionary has been frozen

```
j = t; i := t
**repeat forever**
    **for** k = j +1 **to** i +1 **compute**
        h(k): $x_k \ldots x_{h(k)}$ is the longest match
    **let** k': h(k') is maximum
    $x_j \ldots x_{k'-1}$ is a factor
    j = i+1; i = h(k')
```

Distributed Approximation Scheme

Processors store blocks of length $m(k+2)$ overlapping on $2m$ characters where m is the maximum factor length.

Each processor computes the boundary matches ending furthest to the right.



Distributed Approximation Scheme (2)

Each processor applies the semi-greedy procedure from the first position of the match computed on the left boundary of its block to the position of the match on the right boundary.

Stopping the parsing at the beginning of the match on the right boundary might create for each block a surplus phrase with respect to the optimal parsing of the string (this might happen when the last factor could be extended to cover the right boundary). Therefore, the approximation factor is $(k+1)/k$.

Distributed Approximation Scheme (3)

If $k \geq 10$, there is no relevant loss of compression effectiveness with respect to the optimal factorization.

The approximation scheme is robust on large scale and extreme distributed systems since $m \approx 10$.

Greedy compares to optimal in practice.

If $k \geq 100$, there is no need of computing boundary matches.

The MapReduce Implementation

MR sequence $\mu_0\mu_1\rho_1\mu_2\rho_2\mu_3\rho_3$; $R=2.$

Input Phase $\mu_0\mu_1$ (input string X)

μ_0 maps $(0, X)$ to $U_0 = \{(1, X_1), \ldots (m, X_m)\}$

- $X_i = Y_i\, Z_i$ with $|Y_i| = |Z_i| \geq 300\text{K}$ for $1 \leq i \leq m$

- $Z_i = B_{i,1} \ldots B_{i,r}$ with $|B_{i,j}| = c \geq 1000$ for $1 \leq j \leq r$

μ_1 maps U_0 to $U_0' = \{ (\, i,\ Y_i\,) ,\ (\, (\text{i,j})\ ,\ B_{i,j})$ for $1 \leq i \leq m\ ,\ 1 \leq j \leq r\ \}$

The MapReduce Implementation (2)

MR sequence $\mu_0\mu_1\rho_1\mu_2\rho_2\mu_3\rho_3$

Computational phase $\rho_1\mu_2\rho_2$

<u>first step</u>

ρ_1 reduces U_0' to $U_1 = \{ (i, C_i), (i, D_i), ((i,j), B_{i,j})$ for $1 \le i \le m, 1 \le j \le r \}$.

C_i is the LZW coding of Y_i and D_i is the dictionary learned during the coding process for $1 \le i \le m$.

The MapReduce Implementation (3)

MR sequence $\mu_0\mu_1\rho_1\mu_2\rho_2\mu_3\rho_3$

Computational phase $\rho_1\mu_2\rho_2$

<u>second step</u>

μ_2 maps U_1 to $U_1' = \{\ (\ i,\ C_i\ )\ ,\ (\ (\text{i,j}),\ D_i\ )\ ,\ (\ (\text{i,j}),\ B_{i,j}\ )$ for $1 \leq i \leq m\ ,\ 1 \leq j \leq r\ \}$**.**

ρ_2 reduces ((i,j), $B_{i,j}$) to ((i,j), $C_{i,j}$) with $C_{i,j}$ compressed form of $B_{i,j}$ using D_i as a static dictionary to produce U_2.

The MapReduce Implementation (4)

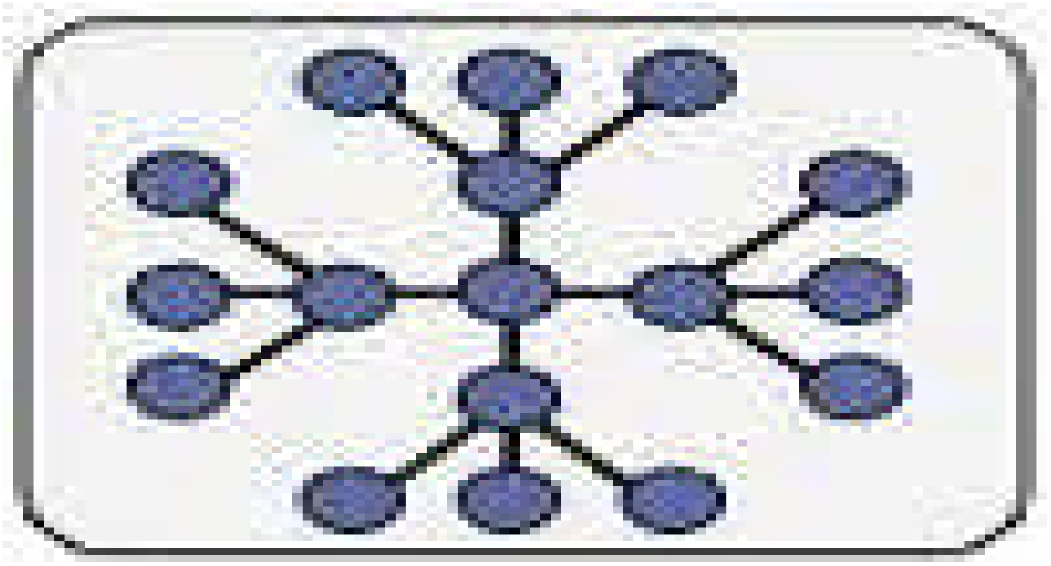
MR sequence $\mu_0\mu_1\rho_1\mu_2\rho_2\mu_3\rho_3$

Output phase $\mu_3\rho_3$

μ_3 maps U_2 to $U_2' = \{(0, C_i), (0, C_{i,j})$ for $1 \le i \le m, 1 \le j \le r\}$.

ρ_3 reduces U_2' to $(0, C = C_1C_{1,1}\ldots C_{1,r}\ldots C_mC_{m,1}\ldots C_{m,r})$ where C is the output .

The Extended Star Network



The central node distributes blocks of 600kB (the first half of 300kB to the adjacent processors and a sub-block of the second half to a leaf).

Robustness and Communication

Low communication cost is required between the two phases of the computation.

If the input phase broadcasts sub-blocks of size between 100 bytes and one kylobyte the computation of boundary matches enhance robustness (extreme case beyond large scale).

Beyond Large Scale

Input Phase $\mu_0\mu_1$ (input string X)

μ_0 maps $(0, X)$ to $U_0 = \{(1, X_1), \ldots (m, X_m)\}$

$X_i = Y_i Z_i$, $Z_i = B_{i,1} \ldots B_{i,r}$ with $100 \leq |B_{i,j}| << 1000$

μ_1 maps U_0 to $U_0' = \{ (i, Y_i) , ((\mathrm{i,j}) , B_{i,j}')$ for $1 \leq i \leq m , 1 \leq j \leq r \}$

$B_{i,j}'$ extends $B_{i,j}$ by overlapping $B_{i-1,j}$ and $B_{i+1,j}$ on m characters.

Compressing Large Size Files

Scalability, robustness and communication are not an issue with SW and LZW compression of very large size files (gigabytes or beyond).

Bounding the dictionary of the LZW compressor with a relaxed version of the *least recently used* strategy has been proposed as the most efficient approach.

De Agostino ICIW 2013

Future Work

The main goal is the design of a robust and scalable approach for lossless compression of arbitrary size files.

Such goal can be achieved experimentally for a specific type of files by decreasing the dictionary size.

A more ambitious project is the design of a new general procedure for distributed memory compression.

Thank you