Advanced Techniques for Combinatorial Algorithms: Data Streams and Map-Reduce

Gianluca Della Vedova

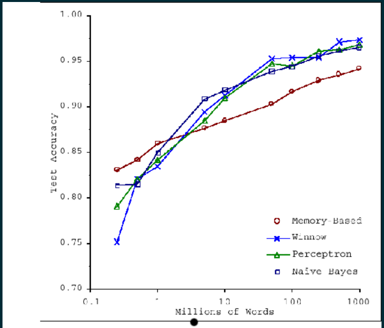
Univ. Milano–Bicocca
https://gianluca.dellavedova.org

May 18, 2020

Gianluca Della Vedova

- Advanced Techniques for Combinatorial Algorithms
- https://gitlab.com/dellavg/advanced-algorithms
- https://gianluca.dellavedova.org
- gianluca.dellavedova@unimib.it

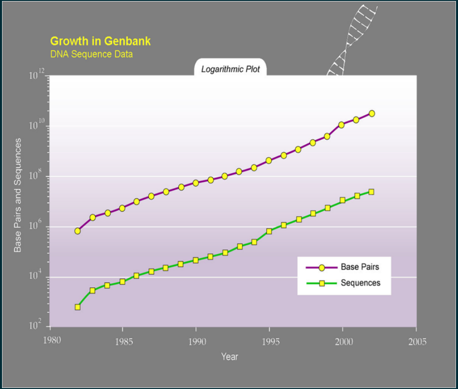
Fact 1



(Banko and Brill, ACL 2001)

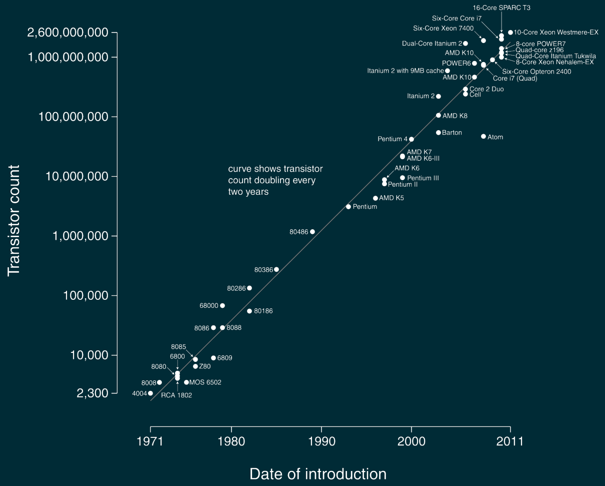
- Huge data are among us

And more are coming



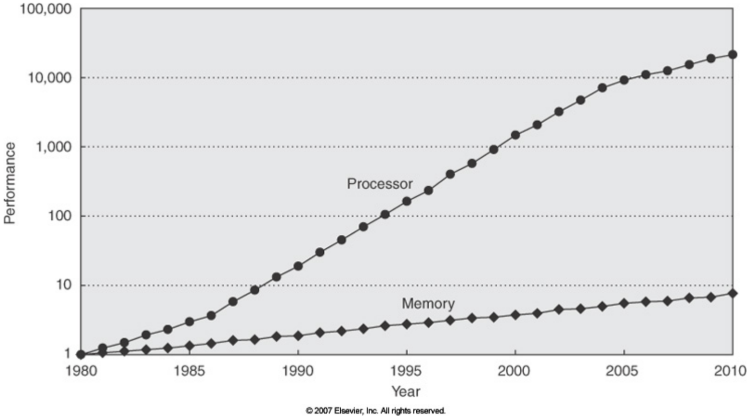
From 1982 to the present, the number of bases in GenBank has doubled approximately every 18 months (`ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt`).

Moore's Law



https://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2011.svg

Fact 2



- Moore's Law is unfair.

Problem

- Input data too large for a single computer
- Don't fit into memory

Solution

- Split data into parts
- If only it were so easy
- Embarassingly parallel problems

Solutions

- parallel algorithms
- map reduce
- data streaming
- External-memory algorithms
- Are all related

Map Reduce model

- Input: hash = <key $\mapsto$ value> pairs
- Map
- Shuffle
- Reduce
- Output: hash

Map Reduce model

- Mapper: receives a $<$key $\mapsto$ value$>$ pair, computes a hash
- Shuffle: all values with same key k are assigned to a unique processor
- Reducer: receives all values with same value k, computes a multiset of $< k, \text{value} >$ pairs

Map Reduce model

- Mapper: receives a <key $\mapsto$ value> pair, computes a hash
- Shuffle: all values with same key k are assigned to a unique processor
- Reducer: receives all values with same value k, computes a multiset of $< k, \text{value} >$ pairs

Map Reduce model

- Mapper: receives a <key $\mapsto$ value> pair, computes a hash
- Shuffle: all values with same key k are assigned to a unique processor
- Reducer: receives all values with same value k, computes a multiset of $< k, \text{value} >$ pairs

Map Reduce model

- Mapper: receives a <key $\mapsto$ value> pair, computes a hash **in parallel**
- Shuffle: all values with same key k are assigned to a unique processor
- Reducer: receives all values with same value k, computes a multiset of $< k, \text{value} >$ pairs

Map Reduce model

- Mapper: receives a <key $\mapsto$ value> pair, computes a hash **in parallel**
- Shuffle: all values with same key k are assigned to a unique processor **automatic**
- Reducer: receives all values with same value k, computes a multiset of $< k, \text{value} >$ pairs

Map Reduce model

- Mapper: receives a $<$key $\mapsto$ value$>$ pair, computes a hash **in parallel**
- Shuffle: all values with same key k are assigned to a unique processor **automatic**
- Reducer: receives all values with same value k, computes a multiset of $< k, \text{value} >$ pairs **sequential algorithm on the values**

k-th Frequency Moment

Instance

k, a list $L = \langle x_1, \ldots, x_n \rangle$ of elements over alphabet Σ

Output

$\sum_{\sigma \in \Sigma} f^k(\sigma)$, where $f(\sigma)$ is the number of occurrences of *sigma* in L

Example

$L = \langle 0, 1, 1, 1, 2, 0, 1, 2, 0, 1, 1, 2 \rangle$

Output, $k = 2$

$3^2 + 6^2 + 3^2 = 54$

Example: k-th frequency moment

Algorithm 1: *k*-FrequencyMoment

Data: k, a list $= L\langle x_1, \dots, x_n\rangle$

$\mu_1(\langle i; x_i\rangle) = \langle x_i; i\rangle$ `/* i is the index */`

$\rho_1(\langle x_i; v_1, \dots, v_m\rangle) = \langle x_i; m^k\rangle$; `/* All occurrences of a symbol are grouped together */`

$\mu_2(\langle x_i; v\rangle) = \langle \$; v\rangle$; `/* Time to sum */`

$\rho_1(\langle \$; v_i, \dots, v_l\rangle) = \langle \$; \sum v_i\rangle$;

Rationale

- No machine can store whole input
- Too many machines = bad
- Shuffling is expensive
- No shared memory
- Loose synchronization

Efficient Algorithm

- at most $O(n^{1-\epsilon})$ machines
- each mapper is a RAM with $O(n^{1-\epsilon})$ space, poly(n) time
- each reducer is a RAM with $O(n^{1-\epsilon})$ space, poly(n) time
- polylogarithmic mapreduce rounds
- reducers have $O(n^{2-2\epsilon})$ overall space
- each key has $O(n^{1-\epsilon})$ values

Classes

- $\mathcal{MRC}^i$ if $O(\log^i n)$ rounds
- $\mathcal{MRC} = \bigcup_{i=0}^{\infty} \mathcal{MRC}^i$
- $\mathcal{MRC}^0$, $\mathcal{MRC}^1$

Map Reduce model

Howard J. Karloff, Siddharth Suri, Sergei Vassilvitskii: A Model of Computation for MapReduce. SODA 2010: 938-948

Relations between models

Map Reduce and PRAM

Reconcile inputs

$x_1, \ldots, x_n \mapsto \langle 1, x_1 \rangle, \ldots, \langle n, x_n \rangle$

Theorem 4.1, Karloff et al, SODA 2010

If $\mathcal{P} \neq \mathcal{NC}$ then $\mathcal{MRC} \not\subseteq \mathcal{NC}$.

Proof

- Pick a $\mathcal{P}$-complete problem, and pad its input
- Use a single reducer to solve it

Map Reduce and PRAM

Theorem 7.1, Karloff et al, SODA 2010

Any CREW PRAM algorithm using $O(n^{2-2\epsilon})$ total memory, $O(n^{2-2\epsilon})$ processors and $t(n)$ time can be run in $O(t(n))$ rounds in $\mathcal{MRC}$.

Proof

- processor $\rightarrow$ reducer
- memory location $\rightarrow$ reducer
- shuffle to coordinate read requests

Map Reduce and PDM

Shuffle step

- Matrix: R=rows=reduce steps, M=columns=keys
- sparse matrix
- Layout depending on maps
- Rearrange into a column-wise ordering

Graph Algorithms

- For each edge e, store its successor $s(e)$
- Euler tour of G = edge-disjoint cycles
- merge any two cycles with a common vertex u

Minimum Spanning Tree (MRC)

Algorithm 2: MST

Data: Array

dense graph $G = \langle V, E \rangle$

/* $|V| = n$, $|E| \geq n^{1+c}$ */

$V_1, \ldots, V_k \leftarrow$ random balanced partition of V;

$E_{i,j} \leftarrow \{(u, v) \in E | u, v \in V_i \cup V_j\}$;

$G_{i,j} \leftarrow$ subgraph of G induced by $E_{i,j}$;

foreach $G_{i,j}$ **do**

  $M_{i,j} \leftarrow$ minimum spanning forest of $G_{i,j}$

$H \leftarrow \left\langle V, \bigcup_{i,j} M_{i,j} \right\rangle$;

$M \leftarrow$ minimum spanning tree of H;

Minimum Spanning Tree (MRC)

MST Algorithm is correct

1. Let $e \in E \setminus E(H)$.
2. Then e is a heaviest edge in some cycles of $G_{i,j}$.
3. The same cycle is also in G.

Minimum Spanning Tree (MRC)

$k = n^{c/2}$. Then $|E_{i,j}|$ is $\tilde{O}(n^{1+c/2})$

1. $W_i = \{v \in V : 2^{i-1} < deg(v) \leq 2^i\}$
2. If $|W_i| < 2n^{c/2} \log n$ then $\sum_{v \in W_i} \deg(v) = \tilde{O}(n^{1+c/2})$
3. Else $|W_i \cap V_j| = O(\log n)$ via Chernoff bound. Proof completed via union bound

Chernoff bound

1. $X = \sum_i^q X_i$, X_i ind. random variables, $Pr[X_i = 1] = Pr[X_i = -1] = 1/2$
2. $Pr[X \geq a] \leq e^{\frac{-a^2}{2q}}$

MRC General Technique

Def. f parallelizable function

1. If there exists functions g, h such that:
2. For any partition $T_i, \ldots, T_k$ of the universe set S, $f(S) = h(g(T_1), \ldots, g(T_k))$
3. g, h are polynomial time computable
4. Output of g has $O(\log n)$ bits.

$\mathcal{MRC}$ algorithm for $f_1, \ldots, f_k$

1. $S_1, \ldots, S_k$ subsets of S, $k \leq n^{2-3\epsilon}$, $\sum |S_i| \leq n^{2-2\epsilon}$
2. $f_1(S_1), \ldots, f_k(S_k)$ with $O(n^{1-\epsilon})$ reducers, $O(n^{1-\epsilon})$ space each.

k-th frequency moment

Algorithm 3: k-FrequencyMoment

Data: k, a sequence $\langle x_1, \ldots, x_n \rangle$

$\mu_1(\langle i; x_i \rangle) = \langle x_i; i \rangle$;
$\rho_1(\langle x_i; v_i, \ldots, v_m \rangle) = \langle x_i; m^k \rangle$;
$\mu_2(\langle x_i; v \rangle) = \langle \$; v \rangle$;
$\rho_1(\langle \$; v_i, \ldots, v_l \rangle) = \langle \$; \sum v_i \rangle$;

k-th frequency moment

Problem: all elements can be the same

$O(n)$ space, instead of $O(n^{1-\epsilon})$

Solution

- $g(\{t_1, \ldots, t_k\}) = k$
- $h(\{i_1, \ldots, i_m\}) = (i_1 + \cdots + i_m)^k$

Proof

- $h(g(T_1), \ldots, g(T_m)) = |S|^k$
- S set of pairs with same value

Maximal Matching (MRC)

Matching

- Let $G = \langle V, E \rangle$ be a graph.
- Then $M \subseteq E$ is a matching if no two edges of M shares a vertex
- M is maximal if $\nexists M_1 \supseteq M$ s.t. M_1 is a matching

Maximal Matching (MRC)

Algorithm 4: MaximalMatching

Data: Array

1 dense graph $G = \langle V, E \rangle$

2 $M \leftarrow \emptyset$, $S = E$;

3 $E' \leftarrow$ random sample of S;

4 **if** $|E'| > \eta$ **then** Fail;

5 $M' \leftarrow$ maximal matching on E' `/* 1 reducer */`

6 $M \leftarrow M \cup M'$;

7 Remove all edges conflicting with M;

8 $S \leftarrow$ remaining edges;

9 **if** $|I| > \eta$ **then** go to step 2;

10 $M' \leftarrow$ maximal matching on M' `/* 1 reducer */`

11 $M \leftarrow M \cup M'$;

Connected components

Algorithm 5: ConnectedComponents

```
Label each vertex either up or down;
foreach edge (v, w) do
    Add the component of the down vertex to that of the up vertex;
    /* The root of the lower component is now a child of the root
       of the upper component                                      */
foreach vertex v do
    parent[v] ← parent[parent[v]]
```

Connected components

```
def map(u, v, p1, p2):
  if p1 == p2: #same component
    return []  #do nothing
  else:
    h1 = hash(p1, r)%2
    h2 = hash(p2, r)%2
    if h1 != h2:
      return [(p1,p2) if h1 else (p2,p1)]

def reduce(list):
  return list[0]
```

Additional Bibliography on Map Reduce

On distributing symmetric streaming computations
`http://portal.acm.org/citation.cfm?doid=1824777.1824786`
Counting triangles and the curse of the last reducer
`http://portal.acm.org/citation.cfm?doid=1963405.1963491`
A Model of Computation for MapReduce `http://dl.acm.org/citation.cfm?id=1873677`

Data Streams

Almost permutation

- Input: a permutation π of $\{1, \ldots, n\}$ with one element missing
- Find the missing element
- Streaming: read the input once
- $\lceil \log n \rceil$ bits of memory

Almost permutation

Algorithm

- Keep the parity bits
- and add $\sum_{i=1}^{n} i$

Reservoir sampling

- Take k elements from a stream
- Add to S the first k elements
- The i-th elements is kept with probability k/i
- In case, remove a random element of S

Uniform sampling

Time t, $i \leq t$. Then $Pr[x_i \in S] = \frac{s}{t}$

Count-Min sketch

- Turnstile model
- Approximate counting: number of occurrences of a symbol
- $d \times w$ matrix $count[]$
- $w = e/\epsilon$, $d = \log 1/\delta$. We want additive error $\leq \epsilon$ with probability $\geq 1 - \delta$
- d hash functions $h_1, \dots h_d : \{1, \dots, N\} \mapsto \{1, \dots, w\}$
- update (j, I_i), then $count[k, h_k(j)] \leftarrow count[k, h_k(j)] + I_i$, $\forall k$
- query $A[i]$, answer $\min_j count[j, h_j(i)]$

HyperLogLog sketch

- Estimate the cardinality of a set of integers
- $p \leftarrow$ Max number of leading zeroes
- Estimate 2^p
- Go through hash function to manage arbitrary sets
- Split into subsets to reduce variance