Divisible Load Scheduling

Henri Casanova$^{1,2}$

$^1$Associate Professor
Department of Information and Computer Science
University of Hawai‘i at Manoa, U.S.A.
$^2$Visiting Associate Professor
National Institute of Informatics, Japan

NII Seminar Series, October 2013
Why are many scheduling problems hard?

- We have seen in the last seminar that many scheduling problems are \( \mathcal{NP} \)-complete.
- It turns out that this is often because of integer constraints.
  - The same reason why bin packing is difficult: you can’t cut boxes into pieces!
- This is somewhat the same idea as the use of *preemption*:
  - \( P||C_{\text{max}} \) is \( \mathcal{NP} \)-complete
  - \( P|\text{pmtn}|C_{\text{max}} \) is in \( \mathcal{P} \)!
- Let’s see this on an example.
$P4 || C_{max}$ example schedule (offline)

$\sum a_i = 21; \; C_{max} = 8$
Let's modify the schedule using preemption.

\[ \sum a_i = 21; C_{\text{max}} = 7 \text{ (optimal: no idle time)} \]
By “cutting” a task in two, we’re able to have all processors finish at the same time
- Zero idle time means the schedule is optimal
- If we were able to cut all tasks into tiny bits, then we would always be able to achieve zero idle time
  - Again, if you have a knife, binpacking is easy
- Question: Can this be done for real-world applications?
It turns out that many useful applications consist of very large numbers of small, independent, and identical tasks.

- task execution time $<<$ application execution time
- tasks can be completed in any order
- tasks all do the same thing, but on different data

**Example applications:**

- Ray tracing (1 task = 1 photon)
- MPEG encoding of a movie (1 task = 1 frame)
- Seismic event processing (1 task = 1 event)
- High-Energy Physics (1 task = 1 particle)

- These applications are termed *Divisible Loads* (DLs)
  - So fine-grain that a *continuous load* assumption is valid

- By the previous seminar, DL scheduling is trivial...
Introduction

Input data?

- In the previous seminar there was no notion of “input data”
  - The implicit assumption was that tasks had access to whatever data they needed
- But in many real-world applications, including DLs, there is some input data for each task
- This input data is stored at some location (the hard drive of a computer)
- If the DL is large, one wants to enroll multiple computers
- Problem: The data must be transferred over the network, which takes time
Introduction

Here comes the network

- When scheduling applications on processors within a single machines (multi-core), one often ignores data transfers (questionable)
- When scheduling applications on distributed platforms, one has to schedule both computation and communication
- Many theoretical scheduling results ignore the network component
  - In some cases communication can be seen as computation, e.g., a computation task depends on a communication task and each type of task can only run on a subset of the “resources”
- Let us define first a very simple execution and platform model...
Master-worker execution

- The computer that holds all input data is called the master ($P_0$)
- All $m$ other computers are called the workers ($P_1, \ldots, P_m$)
- All $P_i$’s can compute (master and workers)
- $P_0$ initially holds $W_{total}$ units of load
- $P_0$ allocates $n_i$ units of load to worked $P_i$
- $\sum_i n_i = W_{total}$
- For now, we completely ignore output data (assume it has size zero)
Bus-shaped platforms

Bus-shaped platform - practice

A bit 1980’s 😊
Bus-shaped platform - theory

- $P_i$ computes one unit of load (one infinitesimal task) in $w_i$ seconds
- $P_0$ sends one unit of load to a worker in $c$ seconds
- $P_0$ can compute and communicate at the same time
- $P_i, i > 0$ must have received all data before beginning computation
  - Questionable assumption, but will make sense with network latencies
- $P_0$ can only communicate with one worker at a time
  - Other versions allow communication to a bounded number of workers
  - We’ll talk about such models in other contexts
- Let’s now draw an example schedule...
Bus-shaped platforms

Example schedule

\[ W_{total} = 6000, \quad c = 1 \]
\[ n_0 = 1000, \quad n_1 = 3000, \quad n_2 = 2000, \quad n_3 = 3000 \]
\[ w_0 = 3, \quad w_1 = 3, \quad w_2 = 5, \quad w_3 = 1.5 \]

**IDLE TIME**
Recursion

- Let’s call $T_i$ the finish time of processor $P_i$
- We can write a recursion with the $T_i$’s, $n_i$’s, $w_i$’s and $c$
- Let’s see it on a picture
Example schedule

- $P_3$
- $P_2$
- $P_1$
- $P_0$

$P_0$: $T_0 = n_0 w_0$
Bus-shaped platforms

Example schedule

\[ P_0 : T_1 = n_1 c + n_1 w_1 \]
Example schedule

\[ P_i : T_i = \sum_{j=1}^{i} n_j c + n_i w_i \]
Given the recursion we have the makespan, $T$, as:

$$T = \max_{0 \leq i \leq m} \left( \sum_{j=0}^{i} n_j c_j + n_i w_i \right)$$

which can be rewritten as:

$$T = n_0 c_0 + \max \left( n_0 w_0, \max_{1 \leq i \leq m} \left( \sum_{j=1}^{i} n_j c_j + n_i w_i \right) \right)$$

which suggests a dynamic programming solution

- An optimal schedule for $m + 1$ processors is constructed from an optimal schedule for $m$ processors
We are stuck

- We now face many difficulties:
  - We don’t have a closed form solution
  - The order of the processors is fixed!
    - We would have to try all $m!$ orders to find the best one
  - The complexity of the dynamic programming solution is $O(W^2_{total}m)$
    - The time to compute the schedule could be longer than the time to execute the application!
  - If we know the optimal schedule for $W_{total} = 1000$, we have to recompute a whole schedule for $W_{total} = 1001$

- Okay, we get it, scheduling is hard 😊
The fact that the $n_i$’s are integers is the root cause of the difficulties.

But in the case of DLs, since $\sum n_i \gg n_i$, a reasonable approximation is to reason on fractions, i.e., rational numbers.

Let $\alpha_i \geq 0$ be the rational fraction of $W_{total}$ allocated to processor $P_i$.

- $n_i = \alpha_i W_{total}$
- $\sum_i \alpha_i = 1$
The DL scheduling approach

- We can now rewrite the recursion in terms of the $\alpha_i$’s
  \[ T = \max_{0 \leq i \leq m} \left( \sum_{j=0}^{i} \alpha_j c + \alpha_i w_i \right) W_{total} \]
- It turns out that with rational $\alpha_i$’s, we can prove two important lemmas

**Lemma (1)**

*In an optimal solution, all processors participate and finish at the same time*

**Lemma (2)**

*If one can choose the master processor, it should be the fastest processor. The order of the worker processors does not matter*
Proof sketch of Lemma 1

- Take some load from the processor that finishes last, give it to another processor (that perhaps does not yet participate)
- Obtain a better schedule, and repeat until all processors finish at the same time
- Let’s see this (informally) on our example schedule...
  - The formal proof is not difficult but not particularly interesting
Bus-shaped platforms

Example schedule
Bus-shaped platforms

Example schedule

$P_0$

$P_1$

$P_2$

$P_3$
Example schedule

Strictly shorter makespan (not optimal)
Proof of Lemma 2

- The master should be the fastest processor, and the order of the workers doesn’t matter.
- In an optimal schedule, we know that $T = T_0 = T_1 = \ldots = T_m$ (Lemma 1).
- Therefore:

  \[ T = \alpha_0 w_0 W_{total}, \]
  \[ T = \alpha_1 (c + w_1) W_{total} \Rightarrow \alpha_1 = \frac{w_0}{c + w_1} \alpha_0, \]
  \[ T = (\alpha_1 c + \alpha_2 (c + w_2)) W_{total} \Rightarrow \alpha_2 = \frac{w_1}{c + w_2} \alpha_1, \]

  \[ \vdots \]

  \[ \Rightarrow \forall i \geq 0 \quad \alpha_i = \prod_{j=1}^{i} \frac{w_j - 1}{c_j + w_j} \alpha_0 \]

  \[ \sum_i \alpha_i = 1 \quad \Rightarrow \quad \alpha_i = \frac{\prod_{j=1}^{i} \frac{w_j - 1}{c_j + w_j}}{\sum_{k=0}^{m} \left( \prod_{j=1}^{k} \frac{w_j - 1}{c_j + w_j} \right)} \]
Proof of Lemma 2

- Let us compute the “work” done in time $T$ by processors $P_i$ and $P_{i+1}$ for $0 \leq i \leq m - 1$

- To ease notations let’s define $c_0 = 0$ and $c_i = c$ for $i > 0$

- We have:

\[ T = T_i = \left( \left( \sum_{j=0}^{i-1} \alpha_j c_j \right) + \alpha_i w_i + \alpha_i c_i \right) W_{total} \]

and

\[ T = T_{i+1} = \left( \left( \sum_{j=0}^{i-1} \alpha_j c_j \right) + \alpha_i c_i + \alpha_{i+1} w_{i+1} + \alpha_{i+1} c_{i+1} \right) W_{total} \]
Proof of Lemma 2

- Let’s define \( K = \frac{T - W_{\text{total}} \left( \sum_{j=0}^{i-1} \alpha_j c_j \right)}{W_{\text{total}}} \)

- We now have \( \alpha_i = \frac{K}{w_i + c_i} \) and \( \alpha_{i+1} = \frac{K - \alpha_i c_i}{w_{i+1} + c_{i+1}} \)

- The total fraction of work processed by \( P_i \) and \( P_{i+1} \) is equal to:
  \[
  \alpha_i + \alpha_{i+1} = \frac{K}{w_i + c_i} + \frac{K}{w_{i+1} + c_{i+1}} - \frac{c_i K}{(w_i + c_i)(w_{i+1} + c_{i+1})}
  \]

- If \( i > 0 \), then \( c_i = c_{i+1} = c \), and the expression above is symmetric in \( w_i \) and \( w_{i+1} \)

- Therefore the order of the workers does not matter
Proof of Lemma 2

- Since $\alpha_i = \frac{K}{w_i+c_i}$ and $\alpha_{i+1} = \frac{K-\alpha_ic_i}{w_{i+1}+c_{i+1}}$

  the total fraction of work processed by $P_0$ and $P_1$ is

  $\alpha_0 + \alpha_1 = \frac{K}{w_0} + \frac{K}{w_1+c}$

- The above is maximized when $w_0$ is smaller than $w_1$

- By induction, we find that it is better to pick the fastest processor as the master
  - Perhaps counter-intuitive?
For Divisible Load applications on bus-shaped networks, in an optimal schedule the fastest computing processor is the master processor, the order of the communications to the workers has no impact on the quality of a solution, and all processors participate and finish simultaneously. The fraction $\alpha_i$ of load allocated to each processor is:

$$\forall i \in \{0, \ldots, p\} \quad \alpha_i = \frac{\prod_{j=1}^{i} \frac{w_{j-1}}{c_j + w_j}}{\sum_{k=0}^{m} \left( \prod_{j=1}^{k} \frac{w_{j-1}}{c_j + w_j} \right)}$$
Star-shaped platforms - practice
Star-shaped platforms - theory

- $P_i$ computes one unit of load (one infinitesimal task) in $w_i$ seconds
- $P_0$ sends one unit of load to worker $P_i$ in $c_i$ seconds
- $P_0$ does not compute (easier to write equations, and no loss of generality as we can add a worker with $c_i = 0$)
Two lemmas revisited

Lemma (1)

*In an optimal schedule all workers participate*

- Simple proof based on the notion of giving some load from the last processor to an unused processor so as to reduce the makespan

Lemma (2)

*There is a unique optimal schedule and in that schedule workers finish at the same time*

- Rather technical proof based on a linear programming formulation and reasoning on the extremal solutions
A third lemma

Lemma (3)

In the optimal schedule the workers are served in non-decreasing order of the $c_i$’s (the $w_i$’s don’t matter!)

Proof: using the same computation as in the proof of Lemma 2 for bus-shaped platforms, for processors $P_i$ and $P_{i+1}$ we have:

$$\alpha_i = \frac{K}{w_i+c_i} \quad \text{and} \quad \alpha_{i+1} = \frac{K-\alpha_ic_i}{w_{i+1}+c_{i+1}}$$

$$\Rightarrow \alpha_i + \alpha_{i+1} = \left(\frac{1}{w_i+c_i} + \frac{1}{w_{i+1}+c_{i+1}}\right) K - \frac{Kc_i}{(w_i+c_i)(w_{i+1}+c_{i+1})}$$

If we exchange $P_i$ and $P_{i+1}$ we obtain:

$$\alpha_i + \alpha_{i+1} = \left(\frac{1}{w_i+c_i} + \frac{1}{w_{i+1}+c_{i+1}}\right) K - \frac{Kc_{i+1}}{(w_i+c_i)(w_{i+1}+c_{i+1})}$$
The difference in processed load between the $P_i, P_{i+1}$ and the $P_{i+1}, P_i$ orders is $\Delta = (c_i - c_{i+1}) \frac{K}{(w_i+c_i)(w_{i+1}+c_{i+1})}$.

The above is not symmetric! Depending on whether $c_i$ is larger/smaller than $c_{i+1}$ the quantity of processed load increases: If $c_{i+1} > c_i$ then $\Delta$ is negative, meaning that the $P_i, P_{i+1}$ order is better than the $P_{i+1}, P_i$ order.

It's easy to verify that communication times are the same in both orders.

Conclusion: more load is processed by serving the workers by non-decreasing $c_i$'s.
For Divisible Load applications on star-shaped networks, in the optimal schedule all workers participate, the workers must be served in non-decreasing $c_i$’s, all workers finish at the same time, and the load fractions are given by:

$$ \alpha_i = \frac{\frac{1}{c_i+w_i} \prod_{k=1}^{i-1} \left( \frac{w_k}{c_k+w_k} \right)}{\sum_{i=1}^{p} \frac{1}{c_i+w_i} \prod_{k=1}^{i-1} \frac{w_k}{c_k+w_k}} $$
So far... so good

- For bus-shaped platforms, we have solved the problem
- For star-shaped platforms, we have solved the problem
- Other have solved it for other platforms shapes (e.g., trees) and variations (e.g., multiple masters)

- A big problem: our model is very naive
- In practice compute costs and communication costs are rarely linear, but *affine*
Latencies

- The time for the master to send $\alpha_i$ units of load to worker $P_i$ is $C_i + c_i \alpha_i W_{total}$
  - e.g., network latency

- The time for worker $P_i$ to compute $\alpha_i$ units of load is $W_i + w_i \alpha_i W_{total}$
  - e.g., overhead to start a process/VM
  - e.g., software overhead to "prepare" the computation
The addition of latencies makes things much harder

The problem is \( \mathcal{NP} \)-complete (even if \( w_i \)’s are zero)
- Non-trivial reduction to 2-PARTITION

All participating workers finish at the same time
- Easy proof

If \( W_{total} \) is large enough then all workers participate and must be served by non-decreasing \( c_i \)’s
- Much more complicated proof

An optimal solution can be found using a mixed linear program...
Linear Programming

- An Integer Linear Program (ILP):
  - A set of integer variables
  - A set of linear constraints
  - A linear objective function

- A Mixed Integer Linear Program (MILP):
  - A set of integer or rational variables
  - A set of linear constraints
  - A linear objective function

- Both (associated decision problems) are \( \mathcal{NP} \)-complete
  - Fully rational Linear Programs can be solved in \( p \)-time!
Linear programming and scheduling

- MILPs occur frequently when formalizing scheduling problems
- Typical integer variables are binary:
  - \( x_{i,j} \): is task \( i \) scheduled on processor \( j \)?
  - \( x_{i,j} \): is the \( i \)-th communication for processor \( j \)?
  - \( \ldots \)
- Typical rational variables:
  - \( \alpha_{i,j} \): the fraction of load processed on processor \( j \)
  - \( \alpha_{i,j} \): the fraction of network bandwidth to processor \( i \) used for task \( i \)
  - \( \ldots \)
Why are MILP formulations useful?

- After all, solving them is \( \mathcal{NP} \)-complete
  - And there may be easy optimal algorithms instead
- Reason #1: provide concise problem description
  - Useful when writing an article
- Reason #2: can be *relaxed* by making all variables rational
  - Solve the rational program in p-time
  - Obtain the (unfeasible) optimal objective function value
  - This value is a *bound on optimal*, which is useful to gauge the quality of heuristics
  - e.g., for a maximization problem: on this instance my heuristic achieves 92, the upper bound on optimal is 100, so I can say my heuristic is (at most) within 8% of optimal.
Mixed Linear Program for DL with latencies

- We define the following variables:
  - $\alpha_i \geq 0$ (rational): $i$-th sent load fraction
  - $y_j$ (binary): true if worker $P_j$ participates
  - $x_{i,j}$ (binary): true if worker $P_j$ received the $i$-th load fraction

- We have the following “setup” constraints:
  - $\sum_i \alpha_i = 1$: the entire load is processed
  - $\forall j \quad \alpha_j \leq y_j$: only participating workers are allocated some load
  - $\forall j \quad \sum_i x_{i,j} = y_j$: a participating worker receives only one fraction of load
Main constraint

With Latencies

The diagram shows the main constraint with latencies.

- First $i-1$ communications
- $i$-th communication and completion
Main constraint

- The time at which the communication of the $i-1$-th load fraction finishes: $\sum_{k=1}^{i-1} \sum_{j=1}^{m} x_{k,j}(C_j + \alpha_j c_j W_{total})$

- The time to communicate and compute the $i$-th load fraction: $\sum_{j=1}^{m} x_{i,j}(C_j + \alpha_j c_j W_{total} + W_j + \alpha_j w_j W_{total})$

- Let $T_f$ be the finish time (of all processors)

- We have the constraint:

  $\forall i \sum_{k=1}^{i-1} \sum_{j=1}^{m} x_{k,j}(C_j + \alpha_j c_j W_{total}) + \sum_{j=1}^{m} x_{i,j}(C_j + \alpha_j c_j W_{total} + W_j + \alpha_j w_j W_{total}) \leq T_f$

- And the objective is to minimize $T_f$
Mixed Linear Program for DL with latencies

**Mixed Integer Linear Program**

minimize $T_f$ subject to

1. $\forall i, 1 \leq i \leq m, \quad \alpha_i \geq 0$
2. $\sum_{i=1}^{m} \alpha_i = 1$
3. $\forall j, 1 \leq j \leq m, \quad y_j \in \{0, 1\}$
4. $\forall i, j, 1 \leq i, j \leq m, \quad x_{i,j} \in \{0, 1\}$
5. $\forall j, 1 \leq i \leq m, \quad \sum_{i=1}^{m} x_{i,j} = y_j$
6. $\forall j, 1 \leq i \leq m, \quad \alpha_j \leq y_j$
7. $\forall i, 1 \leq i \leq m, \quad \sum_{k=1}^{i-1} \sum_{j=1}^{m} x_{k,j} (C_j + \alpha_j c_j W_{total})$
   \[+ \sum_{j=1}^{m} x_{i,j} (C_j + \alpha_j c_j W_{total} + W_j + \alpha_j w_j W_{total}) \leq T_f\]
In everything we’ve seen so far, there are $m$ communications to $m$ workers.

This leads to a lot of idle time, especially if $m$ is large.
Simple idea: get workers to work early
Even better: hide communication (note the homogeneity)
Several variations of this problem have been studied.

Many authors have studied the following question: "Given a number of rounds, how much work should be allocated at each round and how?"

- Worthwhile question with linear or affine models

More interesting: "How many rounds should be used?"

- Linear models: an infinite number of rounds!
  - "Obvious" but long and technical proof
  - Surprisingly not acknowledged in early DL literature

- Affine models: $\mathcal{NP}$-complete

Let’s see the known results for both questions above.
Multi-Round scheduling

Homogeneous bus, given number of rounds

- Assume everything is homogeneous \((c_i = c, C_i = C, w_i = w, W_i = W)\) and the number of rounds is \(M\)
- At each round \(m\) "chunks" are sent, one per worker
- Each chunk corresponds to a fraction \(\alpha_j, 0 \leq j < Mm\)
- For convenience we number these chunks in reverse order
  - The first one is \(Mm - 1\), the last one 0
- Let \(R = w/C\) the computation-communication ratio
- Let \(\gamma_i = \alpha_i w W_{total}\) (the compute time of chunk \(i\))
- Let’s write equations that ensure that there is no idle time
No non-initial idle time

- There is no idle time (after the first round) if a worker computes $X$ seconds and the next $m$ communications also take $X$ seconds.
  - In that way, a worker finishes computing round $j$ right when its chunk for round $j+1$ has arrived!

\[
\forall i \geq m, \quad W + \gamma_i = \frac{1}{R} (\gamma_{i-1} + \gamma_{i-2} + \cdots + \gamma_{i-m}) + mC
\]
All workers finish at the same time

For all workers to finish at the same time, the compute time of the last chunk at a worker should be equal to the time for all remaining communication, and the computation time of the last chunk

\[ \forall 0 \leq i < m, \quad W + \gamma_i = \frac{1}{R} (\gamma_{i-1} + \gamma_{i-2} + \cdots + \gamma_{i-m}) + iC + \gamma_0 \]

To ensure correctness, we also have

\[ \forall i < 0, \quad \gamma_i = 0 \]
\( \forall i \geq m, \quad W + \gamma_i = \frac{1}{R}(\gamma_{i-1} + \gamma_{i-2} + \cdots + \gamma_{i-m}) + mC \)
\( \forall 0 \leq i < m, \quad W + \gamma_i = \frac{1}{R}(\gamma_{i-1} + \gamma_{i-2} + \cdots + \gamma_{i-m}) + iC + \gamma_0 \)
\( \forall i < 0, \quad \gamma_i = 0 \)

- The recursion above corresponds to an infinite \( \gamma_i \) series
- Can be solved using a generating function:
  \[ G(x) = \sum_{i=0}^{\infty} \gamma_i x^i \]
- Using the two recursions above we obtain:
  \[ G(x) = \frac{(\gamma_0 - mC)(1 - x^m) + (mC - W) + C\left(\frac{x(1 - x^{m-1})}{1 - x} - (m - 1)x^m\right)}{(1 - x - x(1 - x^m)/R} \]
- Using the rational expansion theorem, we obtain the roots of the polynomial denominator, and thus the \( \gamma_i \) values!!
Computing the optimal number of rounds is $NP$-complete in the general case (i.e., non-homogeneity).

One "brute-force" option is to do an exhaustive search on the number of rounds, searching for the number of rounds that achieves the lowest makespan.

- Potentially exponential time, but in practice likely very doable.

A more elegant approach consists in writing an equation for the makespan and solving an optimization problem.

- Not difficult (based on a Lagrange Multiplier method).
- Can be extended to heterogeneous platforms.
An interesting theoretical result

Theorem

On any bus- or star-platform, with either linear or affine models, a multi-round schedule cannot improve an optimal single-round schedule by more than a factor 2

Proof:

- Let $S$ be any optimal multi-round schedule, which uses $K$ rounds, and has makespan $T$
- We have $m$ workers, and each received a load fraction $\alpha_i(k)$ at round $k$
- From $S$ we construct a new schedule $S'$ that sends in a single message $\sum_{k=1}^{K} \alpha_i(k)$ to workers $i$
- The master does not communicate more in $S'$ than in $S$ (in fact, less with latencies)
An interesting theoretical result

Therefore, not later than time $T$ all workers have received their load fractions (very coarse upper bound)

- No worker will compute more in $S'$ than in $S$
- Therefore, none of them will spent more than $T$ time units to compute in $S'$
- Conclusion: the makespan of $S'$ is at most $2T$
## Conclusion

### So, what do we know?

<table>
<thead>
<tr>
<th></th>
<th>Bus</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linear</strong></td>
<td>$M = 1$: closed-form</td>
<td>$M = 1$: closed-form</td>
</tr>
<tr>
<td></td>
<td>optimal: $M = \infty$</td>
<td>optimal: $M = \infty$</td>
</tr>
<tr>
<td></td>
<td>given $M &lt; \infty$: closed-form</td>
<td>given $M &lt; \infty$: closed-form</td>
</tr>
<tr>
<td><strong>Affine</strong></td>
<td>$\mathcal{NP}$-complete (1-round MILP)</td>
<td>$\mathcal{NP}$-complete (1-round MILP)</td>
</tr>
<tr>
<td></td>
<td>given $M$, homogeneous: closed-form</td>
<td>optimal $M$: heuristics</td>
</tr>
<tr>
<td></td>
<td>optimal $M$: heuristics</td>
<td></td>
</tr>
</tbody>
</table>

- All processors must finish at the same time
- Multi-round buys at most a factor 2 improvement
- Linear models are strange, but latencies make everything difficult (non-divisible!)
What about sending back results?

- There are essentially no known general results if return messages are to be scheduled.
- If returned messages have the same size as the sent messages, it is easy to come up with the best FIFO (same order) and LIFO (reverse order) strategies.
- But it is easy to find examples in which optimal is neither FIFO nor LIFO.
- Essentially: nobody knows 😊.
Conclusion

Sources and acknowledgments

V. Bharadwaj
D. Ghose
T. Robertazzi

Y. Robert
F. Vivien

H. Casanova
A. Legrand
Y. Robert