

Centrale DIGITAL LAB Le Plateforme

## Big Data

# Performance, efficiency and scalability metrics

Stéphane Vialle & Gianluca Quercini

université PARIS-SACLAY  
CentraleSupélec  
ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

L1SN  
LABORATOIRE NATIONAL DES SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

GrandEst  
Région Île de France

1

## What to do with more computer resources?

Maximal size of a processed problem (*size up*)

Nb of independent problems processed per seconde (*flow strength increase*)

Processing speed of a problem (*speedup*)

Passage à l'échelle / Scaling up : size up + speedup + cost control

2

## 1 – Preliminary analysis of the execution time

- Should performance analysis be pursued?
- Choice of an adapted representation

3

## Should perf. analysis be pursued?

Performance metrics

### Disappointing time curve

When the additional costs of managing parallelism ( $T_{synchro}$ ,  $T_{comm}$ ) are high ...

...the initial acceleration is low!

When the parallel algorithm is much more complex than the best sequential algorithm...

...the final acceleration remains low!

Stop the analysis! Improve the algorithm!

4

## Should perf. analysis be pursued?

Performance metrics

### Promising time curve

When the parallel algorithm has :

- an execution time that decreases significantly
- a limited additional cost on a single resource

All accelerations will be great !

Interesting solution! Continue the analysis

Texec vs Texec ideal  
Speedup  
Efficiency  
Size up  
Scalability...

5

## Choice of an adapted representation

Performance metrics

### Which representation to adopt for an execution time curve?

measures

What is the « best » experimental curves?

- We must compare the measurements to the theory, to the ideal curve
- We need a representation that is easy to verify for human eyes

### Case of a parallel execution time:

- Ideal curve:  $T(n \text{ ressources}) = T(1)/n$  : a hyperbole  
→ Check if we get a hyperbole
- But human eyes do not detect hyperboles  
→ It is easily confused with a curve from another law!

6



Performance metrics

## Acceleration metric

**Cases of hyper-acceleration:**

It's not magic, and it's not normal !  
 → The phenomenon must be analyzed and explained  
 → Correct an error or exploit an optimization

**Examples of explanations :**

- we no longer do the right operations (false result)
- the data fits in the total cache of the P processors
- we have modified the starting algorithm and converge faster (e.g. optimized genetic algorithm!).
- we look for a solution in a tree and stop the pgm

13

Performance metrics

## Efficiency metric

**Efficacité :**

$$e(P) = \frac{S(P)}{P}$$

Resource utilization rate, or fraction obtained of the ideal acceleration

- $e(P) \in [0;1], \in [0\%;100\%]$
- $e(P) > 100\% \Leftrightarrow$  hyper-acceleration

The **user** is interested in the **acceleration** achieved  
 The **buyer** of the machine is interested in the **efficiency** of the applications  
 The **developer** is interested in **both**

14

Performance metrics

## Choice of the sequential reference

**Which sequential execution to choose as a reference?**

- Same program running on a single processor?  
Same algorithm implemented sequentially?  
Best known sequential algorithm ?
- Sequential compilation with the same compiler?  
Compiling with the best sequential compiler ?
- Sequential optimizations allowed by parallelization ?  
Maximum sequential optimizations ?
- Execution on a single processor of the parallel machine?  
Execution on the best sequential machine ?

15

Performance metrics

## Choice of the sequential reference

Each sequential reference choice corresponds to:

- a different point of view
- a different business objective
- a different analysis objective

→ Make the choice corresponding to your problem  
 → Clearly state this choice

**Examples:**

*Final user* → chooses HIS sequential pgm on HIS sequential machine  
*Parallel Code Developer* → chooses HIS parallel pgm in ONE process of HIS parallel machine

16

## 3 – Size Up metrics

- Design of a code suitable for size up
- Size up metrics in execution time
- Size up metrics in time and resources
- Size up in 3 successive objectives
- Experimental examples

17

Size up metrics

## Design of a code suitable for *size up*

**1<sup>st</sup> objective: to be able to deal with bigger problems on more rsracs**

A code that **replicates** most of its data on all machines will always be limited by a machine's memory size...and will **not be suitable for size up**

18

Size up metrics

## Design of a code suitable for *size up*

**1<sup>st</sup> objective: to be able to deal with bigger problems on more rsrcs**

A code that **distributes** most of its data across all machines will be able to store more data on more machines...and **will be suitable for size up**

RAM Code 1 PC → 3 PC

Speedup successful & Size up compliant

First size up criterion passed

19

Size up metrics

## Design of a code suitable for *size up*

**1<sup>st</sup> objective: to be able to deal with bigger problems on more rsrcs**

→ Design of an initial distribution of data, and a minimum volume communication scheme

RAM Code 1 PC → 3 PC

Speedup successful & Size up compliant

First size up criterion passed

- **Need to circulate initial data between nodes:** so that a node can continue calculations on data other than its own.
- **Need to circulate intermediate results between nodes:** so that one node can continue another node's calculations with its own data

20

Size up metrics

## Metric of *size up* in $T_{exec}$

**2<sup>nd</sup> objective: keep  $T_{exec}$  constant**

$T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$   
with:  $T(\text{pb size, rsrc nb})$

Exec. time

$T(100,1)$

$n = 100$  200 400 800

Nb of nodes (log)

Use case complexity:  $O(n^2)$

Ideal size up

Size up maintaining constant exec time

Very bad size up!

→ Goal not achieved!

21

Size up metrics

## Metric of *size up* in $T_{exec}$ and rsrcs

**3<sup>rd</sup> objective: keep  $T_{exec}$  constant, with the minimum number of rsrcs**

$T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$   
with:  $T(\text{pb size, rsrc nb})$   
with:  $O(p_k) = O(\text{calcul}(k \times n_1))$

Exec. time

$T(100,1)$

$n = 100$  200 400 800

Nb of nodes (log)

Use case complexity:  $O(n^2)$

Ideal size up

Right size up

Too expensive size up!

→ Goal not achieved!

→ Goal achieved

22

Size up metrics

## Metric of *size up* in $T_{exec}$ and rsrcs

**3<sup>rd</sup> objective: keep  $T_{exec}$  constant, with the minimum number of rsrcs**

$T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$   
with:  $T(\text{pb size, rsrc nb})$   
with:  $O(p_k) = O(\text{calcul}(k \times n_1))$

$p_k$ : nb of nodes (log)

64 32 16 8 4 2 1

100 200 400 800

n: pb size (log)

Use case complexity:  $O(n^2)$

Ideal size up

Right size up

Too expensive size up!

→ Goal not achieved!

→ Goal achieved

23

Size up metrics

## Size up in 3 successive objectives

**1<sup>st</sup> objective: to be able to deal with bigger problems on more rsrcs**

RAM Code 1 PC → 3 PC

Speedup successful & Size up compliant

First size up criterion passed

**2<sup>nd</sup> objective: keep  $T_{exec} = C^{te}$**

Exec. time

$T(100,1)$

$n = 100$  200 400 800

Nb of nodes (log)

Use case complexity:  $O(n^2)$

Ideal size up

Right size up

Very bad size up!

Size up maintaining constant exec time

→ Goal not achieved!

→ Goal achieved

**3<sup>rd</sup> objective:  $T_{exec} = C^{te}$  with the min nb of rsrc**

$p_k$ : nb of nodes (log)

64 32 16 8 4 2 1

100 200 400 800

n: pb size (log)

Use case complexity:  $O(n^2)$

Ideal size up

Right size up

Too expensive size up!

→ Goal not achieved!

→ Goal achieved

24

Size up metrics

## Experimental example

Co-simulation of heat exchanges within buildings  
weakly coupled (quasi-independent calculations)

**Objective: size up at constant time with minimum nb of rsrc**

$$T(1 \times \text{building}, p_1) = T(2 \times \text{bldg}, p_2) = T(k \times \text{1bldg}, p_k) = C^{te}$$

with:  $O(p_k) = O(\text{calcul}(k \times \text{1bldg})) \approx O(k)$

**A lot of calculations and little comm.** → Size up from 1Gb/s

**Few calculations and little comm.** → Size up from 10Gb/s

25

## 4 – Criteria for scaling

- A metric based on execution time
- *Size Up + Speedup* combined approach
- Scaling graphs

26

Criteria for scaling

## A metric based on T<sub>exec</sub>

As the size of the pb. grows, sequential exec. is no longer possible:

- not enough RAM, not enough disk...
- execution too long, resources cannot be monopolized...

← impossible to execute and measure  $T(n,1)$

- a sequential reference cannot be measured !
- we can't calculate Speedup !!

→ A scaling analysis should be built only on measures of  $T_{exec}$

27

Criteria for scaling

## Size Up & Speedup approach

Definition and criteria for "simple" scaling:

- Enable a "size up", in constant time,
- and in an economically bearable way

Level-3 size up

Exec. Time (log)

Use case complexity:  $O(n^2)$

$T(100,1)$

$n_0 = 100$     $2 \times 100$     $4 \times 100$     $8 \times 100$

Experimental size up

Ideal size up

Nb of computing resources (log)

28

Criteria for scaling

## Size Up & Speedup approach

Definition and criteria for "complete" scaling:

- Enable a "size up", in constant time,
- and in an economically bearable way
- &
- Enable to "speedup" for all problem sizes
- with always the same profile of decrease in execution time

Exec. Time (log)

Use case complexity:  $O(n^2)$

$T(100,1)$

$n_0$     $2n_0$     $4n_0$     $8n_0$

Experimental size up

Ideal size up

Ideal execution time with  $n=n_0$

Experimental execution time with  $n=n_0$

Nb of computing resources (log)

29

Criteria for scaling

## Scaling graph

Creation and use of a scaling graph:

- Measure  $T(n,p)$  for different sizes of pb ( $n$ ) and rsrc ( $p$ )
- Draw  $T(n,p)$  curves in logarithmic scale

**Ideal case:**  
parallel straight lines!

Ideal exec. time (log)

Size up

Speedup

1 building

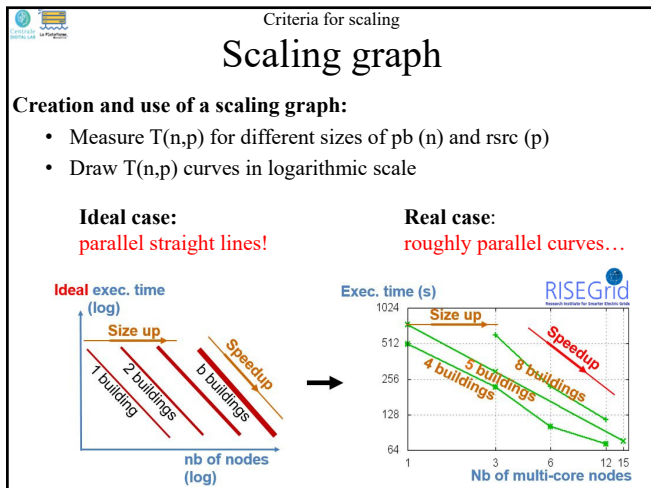
2 buildings

5 buildings

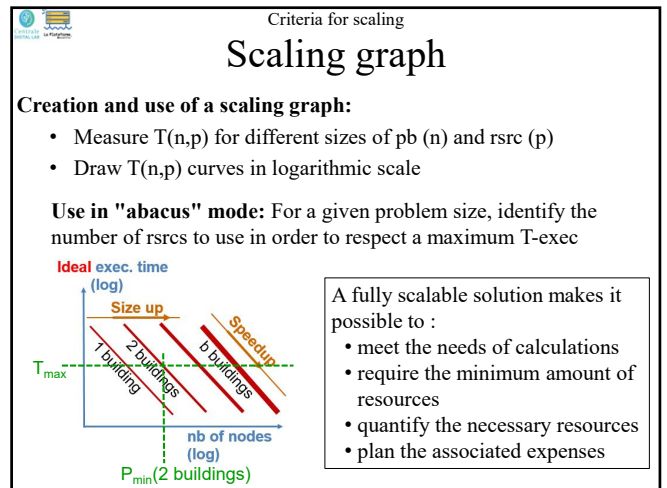
nb of nodes (log)

"To be able to deal with problems of unlimited size in the future"

30



31

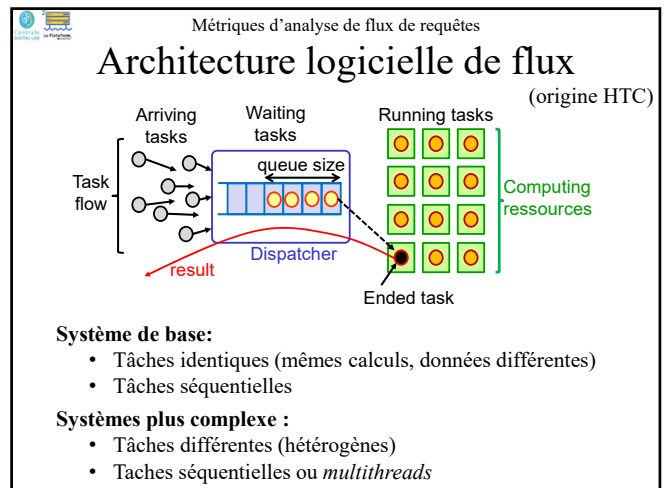


32

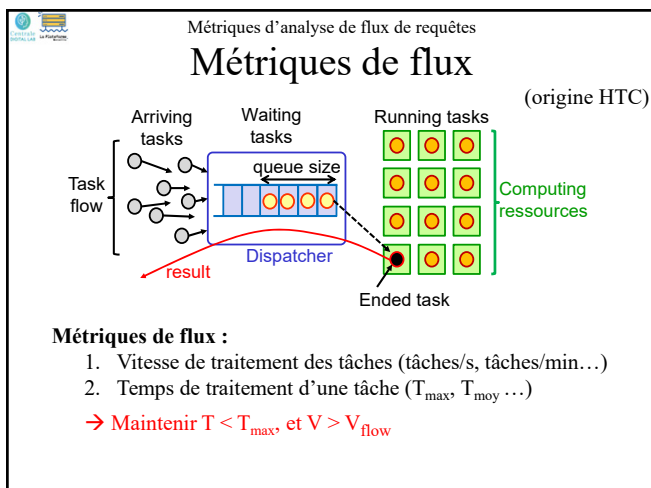
## 5 – Métriques d'analyse de flux de requêtes

- Architecture logicielle de flux
- Métrique de flux
- Analyse de pics de charge

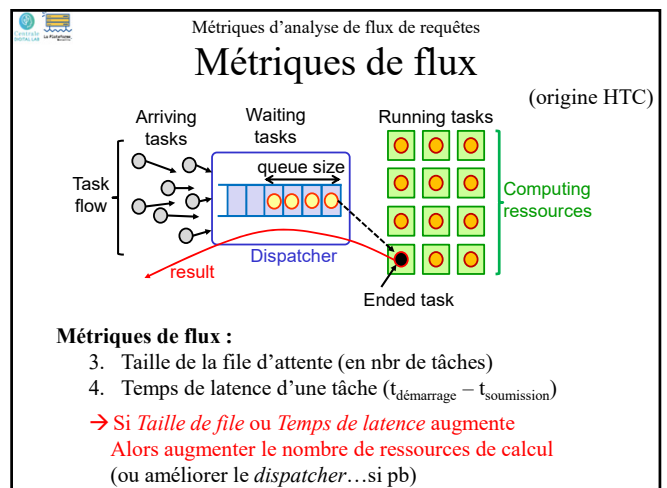
33



34



35



36

Métriques d'analyse de flux de requêtes

## Analyse de pics de charge

(origine HTC)

Arriving tasks → queue size → Dispatcher → Ended task

Waiting tasks → Running tasks → Computing resources

result

**Pour étudier les pics de charge :**

Métrique de moyenne	→	Métrique instantanée
• V	→	V(t)
• T <sub>max</sub>	→	T <sub>max</sub> (t)
• Taille	→	Taille(t)
• T <sub>latence</sub>	→	T <sub>latence</sub> (t)

37

## 6 – Distributed execution of a Spark task graph

- Execution trace
- Performance measures & analysis

38

Distributed execution of a Spark task graph

## Execution trace

**Spark job trace: on 10 Spark executors, with 3GB input file**

```

DAGScheduler: Submitting 24 missing tasks from ShuffleMapStage 0 ...
TaskSchedulerImpl: Adding task set 0.0 with 24 tasks
...
Beginning of task graph execution
TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, 172.20.10.14, executor 0, partition 1, ...)
TaskSetManager: Starting task 2.0 in stage 0.0 (TID 2, 172.20.10.11, executor 7, partition 2, ...)
...
TaskSetManager: Starting task 10.0 in stage 0.0 (TID 10, 172.20.10.11, executor 7, partition 10, ...)
...
TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 18274 ms ... (executor 7) (1/24)
TaskSetManager: Starting task 11.0 in stage 0.0 (TID 11, 172.20.10.7, executor 8, partition 11, ...)
TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 1459 ms ... (executor 8) (2/24)
...
TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
...
End of task graph execution
Submitting a new task when a previous one has finished
  
```

39

Distributed execution of a Spark task graph

## Performance measures & analysis

**Execution time as a function of the number of Spark executors**

Ex. of Spark application run:

- from 1 up to 15 executors
- with 1 executor per node

Good overall decrease but plateaus appear!

**Probable load balancing problem...**

Exec Time(s)

Nb of nodes

Ex: a graph of 4 parallel tasks

on 1 node: T    on 2 nodes: T/2    on 3 nodes: T/2    → A plateau appears

40

## QUIZ

41

## QUIZ

- You need to purchase a distributed application "A" to run on 4 nodes, and you have the following information:
  - "Application A1 has a speedup of 3.9 on 4 nodes
  - "Application A2 has a speedup of 3.0 on 4 nodes
 What do you do?
- You need to purchase a distributed application "A", and you have the following information:
  - "Application A1 processes  $2 \cdot 10^4$  documents/s on 10 nodes
  - "Application A2 processes  $3 \cdot 10^4$  documents/s on 10 nodes
 What do you do if you have 10 nodes?  
 What do you do if you have 20 nodes?

42

**QUIZ**

3. Cite absolute and relative performance metrics
4. What can prevent a system (hard & soft) from scaling up?
5. We consider an application in the form of a graph of very heterogeneous tasks (long treatments and others short)
  - If we increase the number of nodes without increasing the number of tasks (i.e. the size of the pb), will we obtain a good speedup ?
  - Is a "scaling up" still possible?

43

**QUIZ**

6. You need to increase the number of compute nodes to handle larger problems in the same time.  
 Instead, you are offered to scale up all your nodes from 1 to 2 processors, without any other changes in your cluster (cheaper solution...).
  - What should you check before you decide to scale up with this solution?

44

**Appendix**

**A - Methodology for measurement of execution time**

45

Mesure des temps d'exécution  
**Méthodologie de mesures**

Mesures externes :

```
>time myAppli
>/usr/bin/time myAppli
>times myAppli
>timex myAppli
.....
```

12.002u user  
0.128s system  
12.150 total

Nom et fonctionnement variables selon le système utilisé !!

Fréquemment : total > user + system !!

Simple à utiliser  
Pas de modifications des codes sources

Peu précis: ± 0.5s

46

Mesure des temps d'exécution  
**Méthodologie de mesures**

Mesures internes :

```
time ()
clock ()
gettimeofday ()
omp_get_wtime ()
```

→ Compte les clicks d'horloge  
→ Compte le temps écoulé en s

- Toutes ces routines ne sont pas toujours disponibles !
- "gettimeofday" est en général une bonne solution.
- Parfois il existe des outils plus précis pour mesurer de petites durées.

Plus précis que les mesures externes

Besoin de modifier le code source  
Pas toujours totalement portable

47

Mesure des temps d'exécution  
**Méthodologie de mesures**

Précision des outils et des mesures :

123456789012 . 1234567890123456

← Capacité maximale de l'outil de mesure

→ Précision théorique (cf. doc)

← Précision expérimentale, vu la fluctuation des exécutions

- Ne pas tenir compte de trop de décimales!
- Faire attention à ne pas déborder la capacité de mesure!

48



Mesure des temps d'exécution

## Méthodologie de mesures

**Problème fréquent :**

Test en mode exclusif (mono-user).  
Outil de mesure à 1ms de précision.

→ Fluctuation de 500ms d'une exécution à l'autre !!  
Et plus encore avec la montée en fréquence automatique des procs. (effet de "chauffe")

**Démarche conseillée :**

- Mesurer les fluctuations, ne pas les ignorer (le *warm up* des processeurs peut perturber les premières)
- Ne pas donner que les valeurs moyennes
- Mesurer des temps > 10s si possible

49

Mesure des temps d'exécution

## Méthodologie de mesures

**Stocker des meta-données sur les conditions de mesure :**


- Date de l'exécution**
- Auteur(s) du test**
- Outil(s) de mesure utilisé(s)
- Caractéristiques de la machine : RAM, Cache, Processeurs, ...
- OS utilisé (nom et version)
- Compilateur utilisé (nom et version)
- Options de compilation utilisées
- Test en multi-user/mono-user ?
- Présence d'IO dans le test ?
- Configuration du programme de test : taille des données, ...

On oublie souvent (et rapidement) à quel benchmark se réfère une série de mesures !

On manque souvent de détail sur les conditions de réalisation d'une série de mesures !

50

Performance, efficiency and scalability metrics



51