


CentraleSupélec  

Big Data : Informatique pour les données et calculs massifs

3 – Métriques de performance et de passage à l'échelle

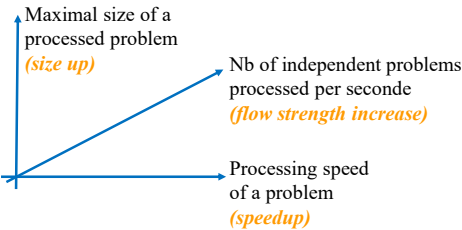
Stéphane Vialle

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

Stephane.Vialle@centralesupelec.fr
<http://www.metz.supelec.fr/~vialle>

CentraleSupélec

Que faire avec plus de ressources informatiques ?



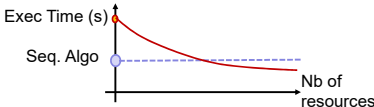
« Passage à l'échelle » : size up + speedup + maîtrise des coûts

CentraleSupélec

Accélération d'un traitement

Difficultés pour accélérer une application (sur plusieurs rsrcs de calcul) :

- Les parties de son code les plus gourmandes en temps doivent être décomposables en sous-tâches pouvant s'exécuter en parallèle
 Certains algorithmes sont intrinsèquement séquentiels !
 → chercher une nouvelle modélisation du problème !!
- L'algorithme parallèle doit avoir une complexité identique ou pas beaucoup plus grande que le meilleur algorithme séquentiel
 Sinon le gain peut être faible !

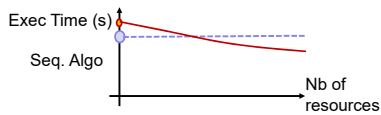


Accélération d'un traitement

Difficultés pour accélérer une application (sur plusieurs rsrce de calcul) :

3. *Le surcoût de gestion de la décomposition en sous-tâches ne doit pas être trop important*

Coût de gestion du parallélisme : temps de synchronisation et temps de communications

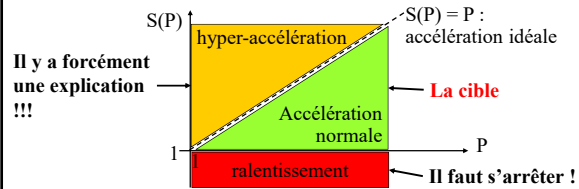


Il y a plein de raisons d'échouer dans une parallélisation !

Accélération d'un traitement

Métrique d'accélération : Speedup sur P ressources

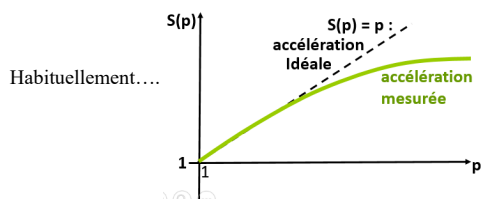
$$S(P) = \frac{T(1)}{T(P)} \rightarrow \begin{cases} S(P) < 1 : \text{on ralentit !} \\ \text{mauvaise parallélisation} \\ 1 < S(P) < P : \text{"normal"} \\ P < S(P) : \text{hyper-accélération} \\ \text{analyser \& justifier} \end{cases}$$



Accélération d'un traitement

Métrique d'accélération : Speedup sur P ressources

$$S(P) = \frac{T(1)}{T(P)} \rightarrow \begin{cases} S(P) < 1 : \text{on ralentit !} \\ \text{mauvaise parallélisation} \\ 1 < S(P) < P : \text{"normal"} \\ P < S(P) : \text{hyper-accélération} \\ \text{analyser \& justifier} \end{cases}$$



CentralesSupélec

Accélération d'un traitement

Efficacité :

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

Taux d'utilisation des ressources, ou fraction obtenue de l'accélération idéale

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

L'utilisateur s'intéresse à l'accélération obtenue

L'acheteur de la machine s'intéresse à l'efficacité des applications exécutées

Le développeur s'intéresse aux deux

CentralesSupélec

Traitement de plus gros pbs

Difficultés à traiter de plus gros problèmes (origine HPC)

Un code qui **réplique** la plupart de ses données sur toutes les machines sera toujours limité par la taille mémoire d'une machine...
...et ne sera **pas apte au size up**

CentralesSupélec

Traitement de plus gros pbs

Difficultés à traiter de plus gros problèmes (origine HPC)

Un code qui **répartit** la plupart de ses données sur toutes les machines pourra stocker plus de données sur plus de machines...
... et sera **apte au size up**

Traitement de plus gros pbs

Difficultés à traiter de plus gros problèmes (origine HPC)

Un algorithme distribué avec répartition des données est souvent complexe :

- *Besoin de faire circuler les données initiales entre les nœuds de calcul* : pour qu'un nœud puisse poursuivre ses calculs sur d'autres données que les siennes
- *Besoin de faire circuler les résultats intermédiaires entre les nœuds* : pour qu'un nœud puisse poursuivre les calculs d'un autre, avec ses données

→ Conception d'une répartition initiale des données, et d'un schéma de communication (à volume minimal...) : **nécessite un expert !**

The diagram illustrates the flow of data from RAM to Code across multiple nodes. It shows a box labeled 'Data' in RAM and a box labeled 'Code' in RAM, with an arrow pointing to three nodes, each with its own 'Code' box. This represents the distribution of code and data to multiple processing units.

Traitement de plus gros pbs

Métrique de size up (extensibilité) : courbe de temps

1 - Maintenir constant le temps d'exécution
 $T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$
 avec : T (taille pb, nb rsrc)

→ **Objectif pas toujours atteignable !**

The graph plots execution time against the number of nodes (log scale). It shows three curves: 'Very bad size up' (red, increasing), 'Successful experimental size up' (green, flat), and 'Ideal size up' (blue, decreasing). The x-axis is labeled 'Nb of nodes' with values 1, 2, 4, 8, 16, 32, 64 (log). The y-axis is 'Exec. time' with a point T(100,1) at n=100.

Traitement de plus gros pbs

Métrique de size up (extensibilité) : courbe de temps

2 - Maintenir constant le temps d'exéc. avec le bon nombre de rsrc
 $T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$
 avec : T (taille pb, nb rsrc)
 Avec $O(p_k) = O(\text{calcul}(n))$

→ **Objectif pas toujours atteignable !**

The graph plots execution time against the number of nodes (log scale) for a case with complexity O(n^2). It shows three curves: 'Successful experimental size up' (green, flat), 'Too expensive size up' (orange, increasing), and 'Ideal size up' (blue, decreasing). The x-axis is labeled 'Nb of nodes' with values 1, 2, 4, 8, 16, 32, 64 (log). The y-axis is 'Exec. time' with a point T(100,1) at n=100. A note says 'Use case complexity: O(n^2)'.

Traitement de plus gros pbs

Métrique de *size up* (extensibilité) : **courbe de coût en rsrc**

2 - Maintenir constant le temps d'exéc. avec le bon nombre de rsrc

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

→ Objectif pas toujours atteignable !

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

Traitement de plus gros pbs

Métrique de *size up* (extensibilité) :

2 - Maintenir constant le temps d'exéc. avec le bon nombre de rsrc

→ Exemple expérimental de courbes de temps

Un problème avec **bcp de calculs** et peu de communications → Size up avec 1Gb/s

Un problème avec **peu de calculs** et peu de communications → Presque un size up avec 10Gb/s

Traitement de flux de requêtes plus intenses

(origine HTC)

Système de base:

- Tâches identiques (mêmes calculs, données différentes)
- Tâches séquentielles

Systèmes plus complexe :

- Tâches différentes (hétérogènes)
- Tâches séquentielles ou *multithreads*

Traitement de flux de requêtes plus intenses (origine HTC)

Métriques *size up* :

1. Vitesse de traitement des tâches (tâches/s, tâches/min...)
2. Temps de traitement d'une tâche (T_{max} , T_{moy} ...)

→ Maintenir $T < T_{max}$, et $V > V_{flow}$

Traitement de flux de requêtes plus intenses (origine HTC)

Métriques de *size up* :

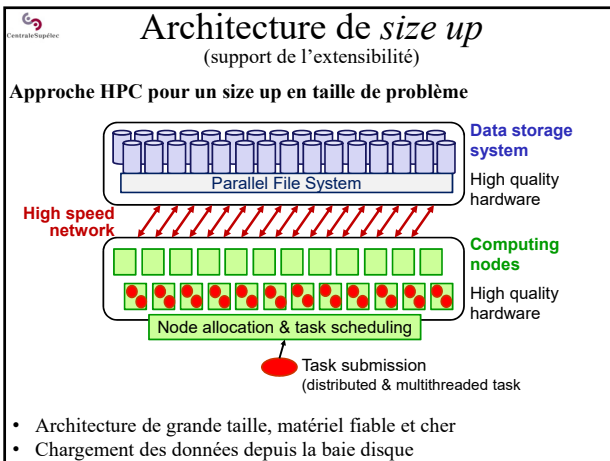
3. Taille de la file d'attente (en nbr de tâches)
4. Temps de latence d'une tâche ($t_{démarrage} - t_{soumission}$)

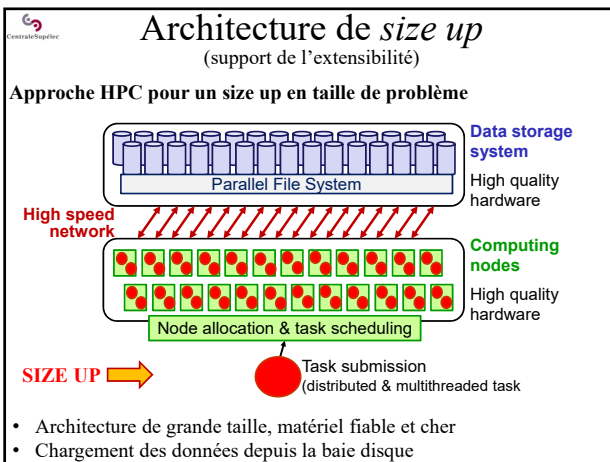
→ Si *Taille de file* ou *Temps de latence* augmente
Alors augmenter le nombre de ressources de calcul
(ou améliorer le *dispatcher*... si pb)

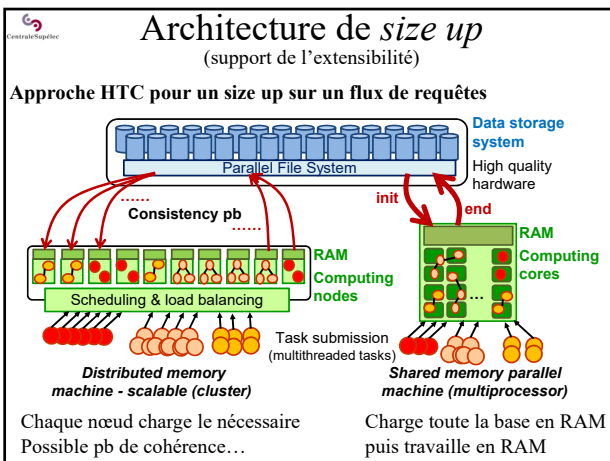
Traitement de flux de requêtes plus intenses (origine HTC)

Analyse des pics de charge :

Métrique de moyenne	→	Métrique instantannée
• V	→	$V(t)$
• T_{max}	→	$T_{max}(t)$
• Taille	→	Taille(t)
• $T_{latence}$	→	$T_{latence}(t)$







CentralesSupélec

Architecture de *size up*

(support de l'extensibilité)

Architecture **Big Data** pour un *size up* en taille de données à traiter

Network of data nodes
Commodity hardware

Job scheduling, load balancing, job management

- Architecture très extensible
- Matériel standard (pas trop cher), moyennement fiable
- Tolérance aux pannes par redondance matérielle et logicielles...

CentralesSupélec

Architecture de *size up*

(support de l'extensibilité)

Architecture **Big Data** pour un *size up* en taille de données à traiter

Network of data nodes
Commodity hardware

Job scheduling, load balancing, job management

Task submission

Data nodes host computing tasks related to their local data

- Architecture très extensible
- Matériel standard (pas trop cher), moyennement fiable
- Tolérance aux pannes par redondance matérielle et logicielles...
- On amène les calculs sur les nœuds possédant les bonnes données

CentralesSupélec

Architecture de *size up*

(support de l'extensibilité)

Architecture **Big Data** pour un *size up* en taille de données à traiter

Network of data nodes
Commodity hardware

Job scheduling, load balancing, job management

Task submission

Data nodes host computing tasks related to their local data

- Architecture très extensible
- Matériel standard (pas trop cher), moyennement fiable
- Tolérance aux pannes par redondance matérielle et logicielles...
- On amène les calculs sur les nœuds possédant les bonnes données

Architecture de *size up*
(support de l'extensibilité)

Architecture **Big Data** pour un *size up* en taille de données à traiter

Network of data nodes
Commodity hardware

Job scheduling, load balancing, job management

Task submission

- Architecture très extensible
- **Matériel standard** (pas trop cher), moyennement fiable
- Tolérance aux pannes par redondance matérielle et logicielles...
- **Beaucoup d'architectures Big Data dans des « clouds »**

Passage à l'échelle en taille de pb

Définition : Prolongement d'une démarche de *size up*

- Supporter le *size up* avec une maîtrise des coûts en ressources
- Reproduire le profil d'accélération pour chaque taille de pb

Exec. Time (log)

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

$n_0 = 100$ 2×100 4×100 8×100

Experimental size up

Ideal size up

Nb of computing resources (log)

Ideal execution time with $n = n_0$

Experimental execution time with $n = n_0$

Passage à l'échelle en taille de pb

Quand la taille du problème augmente :

On ne peut plus réaliser une exécution séquentielle, ou même sur un seul nœud ! (pas assez de RAM, exécution trop longue...)

- on ne peut pas mesurer de référence séquentielle
- on ne peut pas calculer de *Speedup*

Il faut définir une nouvelle métrique pour qualifier la qualité d'un passage à l'échelle en taille de problème

T_a

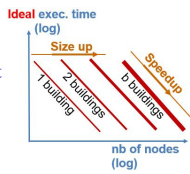
0

p_a

rsrc

Métrie de *passage à l'échelle*

- Pouvoir utiliser efficacement plus de rsrc pour traiter des problèmes plus gros
- Pouvoir maintenir le temps d'exécution est quand la taille du problème augmente
- Pouvoir utiliser seulement le nombre de ressources théoriquement nécessaires



+ Avoir une solution logicielle extensible et déployable sur une architecture matérielle extensible

→ **Pouvoir traiter des problèmes sans limite de taille**

Google, Amazon, Facebook ... ont des problèmes et des solutions « *web scale* » sur des architectures énormes.

Beaucoup d'autres ont des problèmes plus petits mais avec le même souci de passage à l'échelle : ne pas avoir de limites dans le futur

Métriques de performance et de passage à l'échelle