ASSOULINE Daniel
GIROUX Baptiste

Make distribué

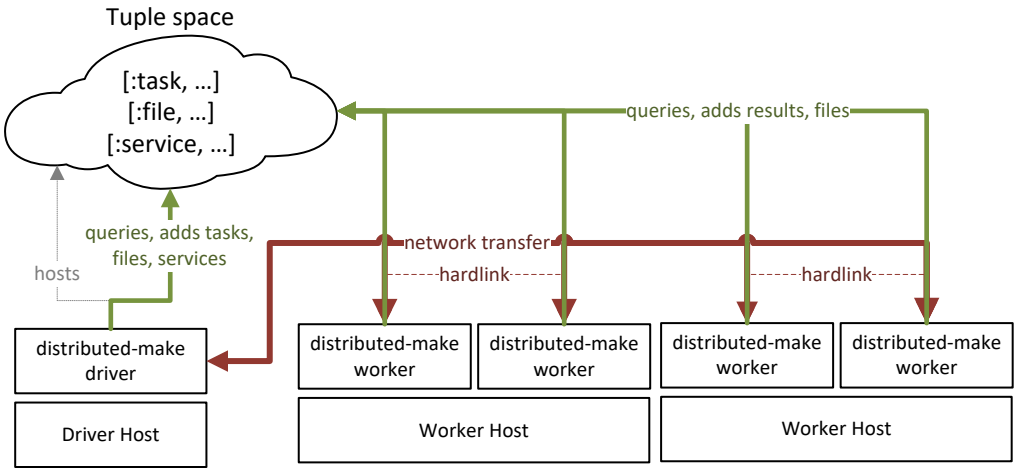
dRuby – 25 novembre 2016

SIBILLE Gaspard
TAVERNIER Vincent

Introduction

Ce document fournit des informations sur l'algorithme utilisé ainsi que les mesures de performances. De plus amples détails sont disponibles dans la documentation du code, qui peut être lue dans un navigateur avec YARD, tel que décrit dans le README.

Architecture et terminologie



- Driver : processus lancé par l'utilisateur, responsable de
  - Parser le Makefile
  - Distribuer les tâches via le tuple space
  - Rassembler les résultats
- Driver host : machine hébergeant le processus driver. Peut aussi héberger des workers si nécessaire.
- Worker : démon pouvant exécuter des tâches récupérées depuis le tuple space
- Worker host : machine hébergeant des processus workers. Dans notre cas, un processus worker est créé par cœur des machines.
- Tuple space : espace d'objets partagé qui supporte les lectures, écritures et suppressions concurrentes atomiques

Algorithme

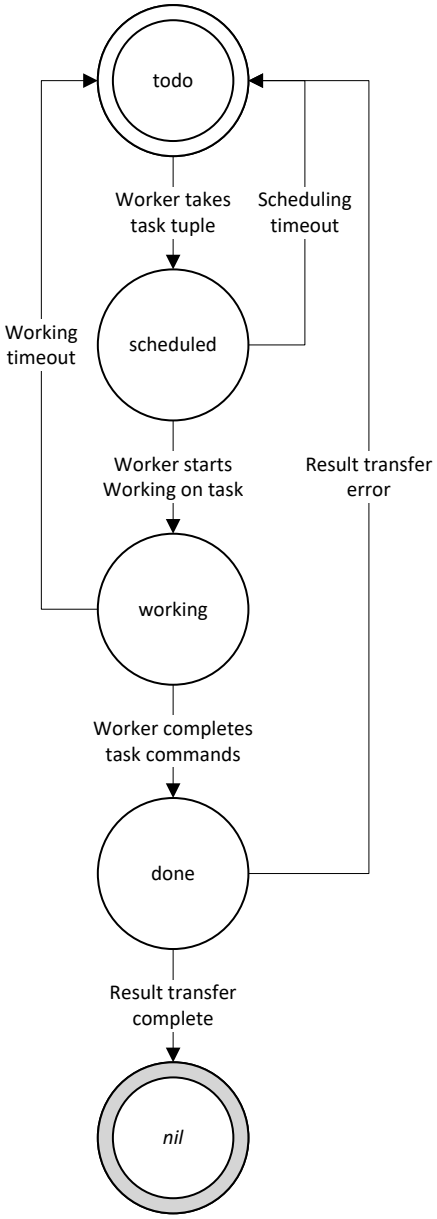
Le tuple space est utilisé pour représenter l'état courant des tâches du Makefile en cours de compilation. Les tuples représentant les tâches sont de la forme [:task, task_name, status] où task_name est le nom de la règle associée, et status l'un des états visibles dans l'automate des tâches page suivante.

Le driver itère les nœuds de l'arbre des dépendances en partant des feuilles de façon à détecter les premières tâches pouvant être réalisées (dépendances disponibles) mais encore réalisables (fichier résultat manquant), et les publie dans le tuple space.

Les workers qui sont en attente de tâche vont se répartir les tâches alors disponibles, en modifiant leur état via des opérations d'écriture sur le tuple space. La durée d'expiration des tuples permet de mettre en place un système de renouvellement qui assure qu'un worker subissant une panne ne

bloque pas l'intégralité du système. La librairie Rinda garantit que les tuples ayant une durée d'expiration expireront lors de l'exécution de la boucle d'expiration.

Les fichiers sont quant à eux représentés par des tuples [:file, file_name, host, is_worker?, handle] qui permettent aux différents agents de rechercher les fichiers sur la même machine (pour éviter un transfert réseau et donc optimiser les performances en cas de localité des dépendances), ou préférer effectuer un transfert de fichier depuis un des workers plutôt que depuis le driver (pour éviter de surcharger le processus responsable de la gestion du tuple space). « handle » est une référence à un objet distribué qui permet le transfert du contenu du fichier via dRuby.



Make distribué

dRuby – 25 novembre 2016

Déroulement des tests

3 séries de mesures ont été effectuées sur les 3 clusters de Grenoble à savoir "Génépi", "Adonis", "Eden". Une série sur Génépi et les deux autres sur une combinaison aléatoire des trois. Les temps sont mesurés à partir du moment où le nombre requis de workers se sont connectés au **tuple space**.

Chaque makefile est alors compilé avec des un nombre de workers allant de 1 à 119. Sachant que les machines utilisées ont 8 cœurs on déploie jusqu'à 8 workers par machine (1 worker par cœur). C'est-à-dire qu'on utilise de 1 à 15 machines. Un cœur sur les 120 disponibles est réservé pour l'exécution du processus driver.

Analyse des résultats

premier(-small)

premier et premier-small sont constitués d'une succession de tâches nécessitant du calcul mais peu de transferts. Ainsi on observe dans les deux cas qu'on atteint un plateau lors de nos mesures. En effet ces Makefiles ne définissent que 20 tâches parallélisables, ajouter plus de 20 workers est superflu.

blender-2.xx

La différence des Makefile blender-2.xx(-recurse) avec premier réside dans la quantité de données qui sont transférées après chaque tâche. En effet, lorsque deux workers d'une même machine ont besoin d'une dépendance donnée, celle-ci est échangée via un **hard-link** donc l'échange est instantané. Mais si les deux workers sont sur deux machines différentes alors l'échange doit se faire via la couche réseau. Donc, lorsque l'on augmente le nombre de machines, on augmente la probabilité de devoir s'échanger l'information par la couche réseau. Le bénéfice obtenu au niveau de la puissance de calcul distribuée est perdu lors du transfert des fichiers produits.

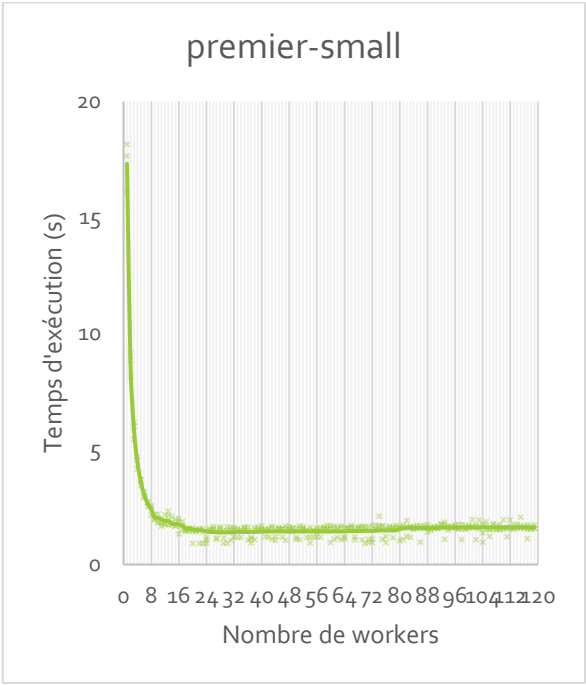
Une possibilité pour résoudre ce problème serait de grouper les différents nœuds de l'arbre des dépendances afin de distribuer aux groupes de workers d'une même machine un sous-arbre dont les règles bénéficieraient de la localité des données.

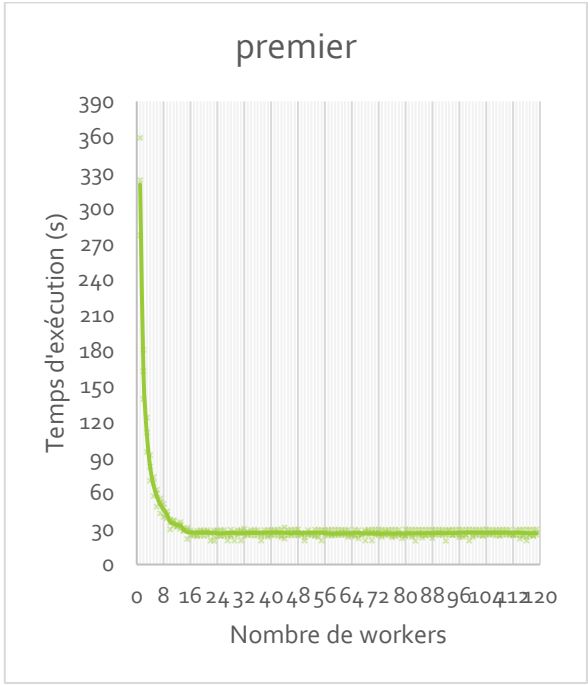
Make distribué

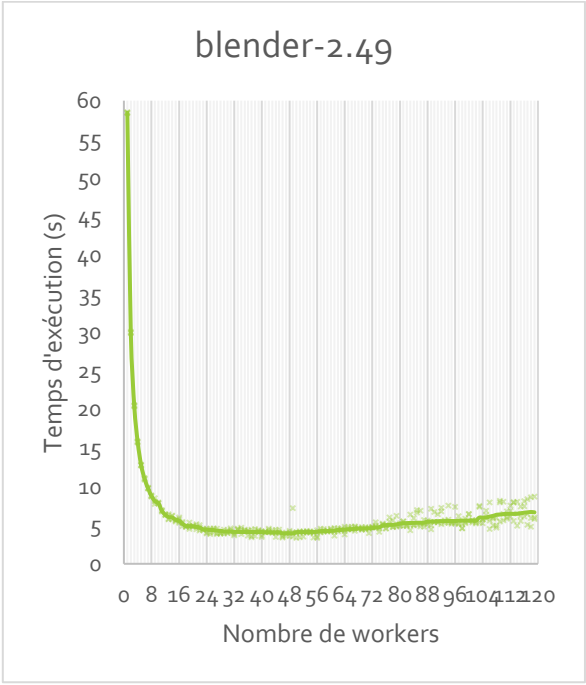
dRuby – 25 novembre 2016

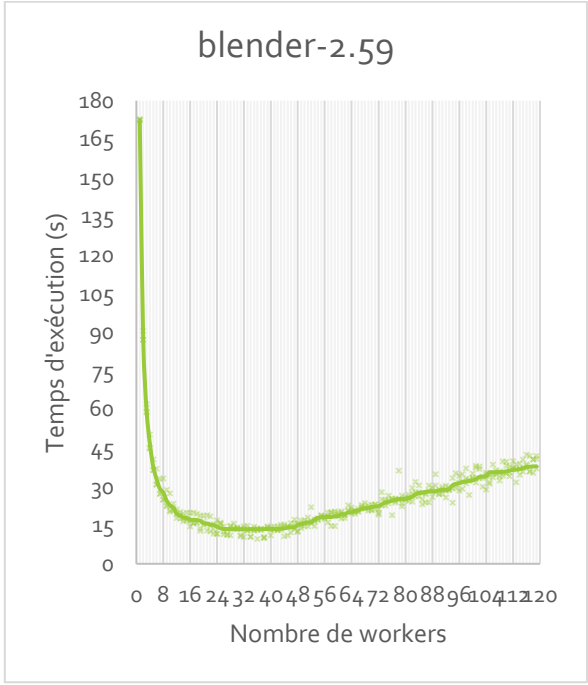
Mesures

Temps d'exécution





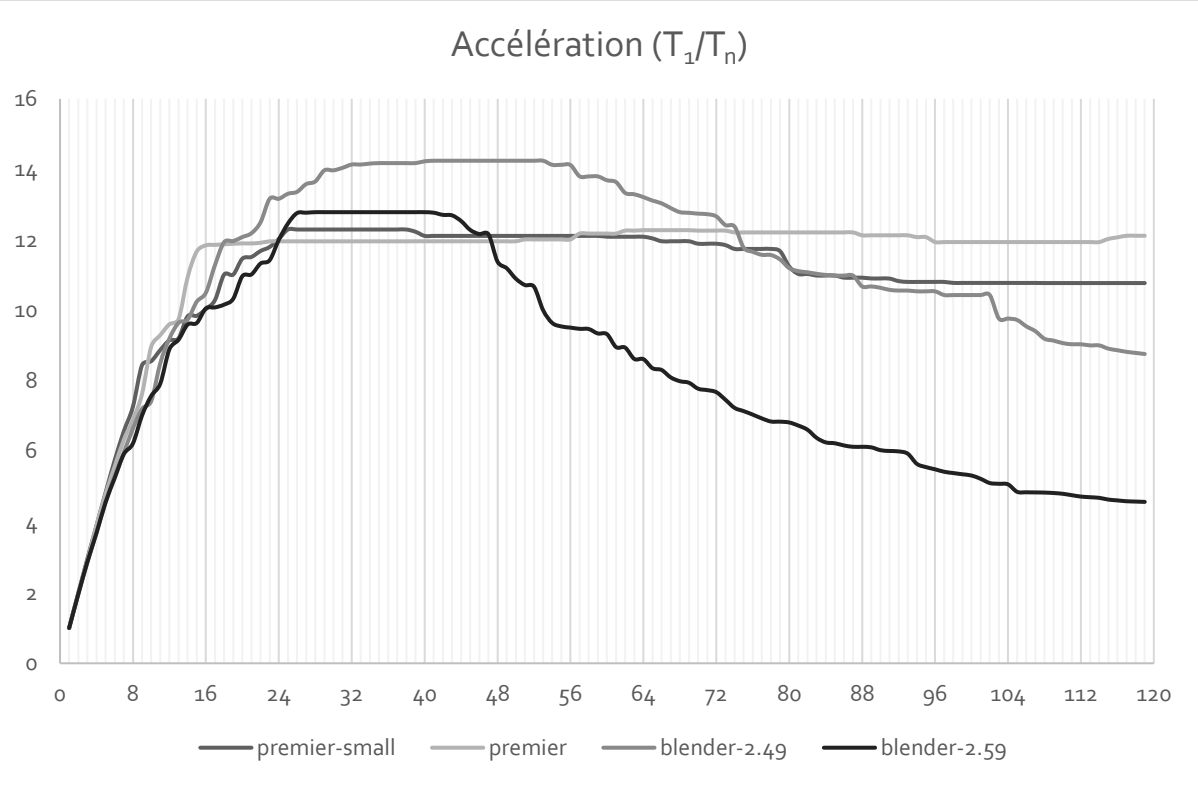




Make distribué

dRuby – 25 novembre 2016

Accélération



Travail