Environnement Canada Environment Canada

Canada

Introduction au datamover

Datamover
CMC, Dorval
Alain St-Denis
CHP et opérations nationales
15 mars 2011

Synopsis

- Qu'est-ce que le datamover?
- Pourquoi le datamover?
- Exemples de commandes non-optimales
- L'API
- Environnement utilisateur
- Exemples d'utilisation
- Le serveur d'exécution
- Questions?

Qu'est-ce que le datamover?

Le datamover est un système de transfert de fichiers asynchrone haute performance.

Ses principaux composants sont un API (implémenté en C, Fortran et shell) et un serveur d'exécution :

- *L'API est simple et facile à utiliser et sert à céduler les tâches de transfert de fichiers.*
- *Le serveur d'exécution optimise les chemins réseau de manière à utiliser les plus performants.*

Pourquoi le datamover?

L'idée du datamover a germé quelque part en 2009 pendant les réunions du comité pour l'intégration du système de suites de tâches (ISST).

Ce système essaie de répondre à deux besoins principaux :

- *Optimiser l'utilisation des processeurs de CHP en minimisant le temps passé en I/O.*
- *Isoler l'utilisateur des complexités inhérentes à l'utilisation optimale des systèmes de fichiers et des chemins réseau utilisés pour effectuer les transferts de fichier.*

Exemples de commandes non-optimales

Note : non-optimales != incorrectes

- À partir de alef :

```
scp /data/cmda/xxx datasvr:/cnfs/dev/yyy
cp /cnfs/dev/xxx /home/afsg/yyy
scp /cnfs/dev/xxx zeta:/fs/dev/yyy
```

- À partir de erg :

```
cp /data/cmdn/xxx /data/cmdn/yyy
```

- À partir d'une job dans la classe development sur zeta :

```
scp /fs/dev/GROS_FICHIER datasvr:/cnfs/dev
```

L'API

L'API est constitué d'une libraire partagée écrite en C. L'API est « thread safe ».

Le rôle de l'API est de préparer une *tâche* qui sera traitée par le serveur d'exécution. La description de la tâche est enregistrée dans un fichier sous le répertoire $HOME/.dmv.

L'API (2)

L'interface implémente les fonctions suivantes :

(*note : pour l'API Fortran, remplacer* _c *par* _f)

- dmv_init_c()

  Initialise le fichier de description de tâche.

- dmv_step_c(char *success_cmd, char *error_cmd)

  Une tâche peut être divisée en *steps*. L'ordre d'exécution des steps n'est pas déterminé. Les paramètres définissent une commande shell à être executée en cas de succès ou d'erreur, respectivement.

- dmv_wait_c()

  Sert à contrôler l'ordre d'exéction des steps.

- dmv_exec_c(char *cmd)

  Exécution d'une commande. Les ressources allouées sont limitées.

L'API (3)

- dmv_launch_c()

  Finalise le fichier de description de tâche et notifie le serveur d'exécution qu'une tâche est prête.

- dmv_copy_c(char *options, char *src, char *dst)

  Copie de fichiers/répertoires. Les options disponibles sont les mêmes que celle supportées par la commande effectuant le transfert (srcp pour notre site).

  Les sources multiples ne sont pas supportées. Les wild cards le sont.

- dmv_move_c(char *options, char *src, char *dst)

  Déplacement de fichiers/répertoires. Effectivement une copie suivie d'une suppression. Les restrictions de dmv_copy_c() s'appliquent.

- dmv_sync_c(char *options, char *src, char *dst)

  Synchronisation de fichiers/répertoires. Utilise rsync. Mêmes restrictions que pour dmv_copy_c().

L'API shell

L'API est aussi implémenté au niveau shell. Toutes les fonctions décrites précédemment ont leur équivalent :

- dmvinit
- dmvstep
- dmvwait
- dmvexec
- dmvlaunch
- dmvcp
- dmvmv
- dmvsync

Contrairement aux API C (et Fortran) qui doivent commencer une tâche avec un appel à dmv_init_c() et terminer avec dmv_launch_c(), ils peuvent ici être omis si dmvstep ou dmvwait ne sont pas utilisés.

Environnement utilisateur

- Accès au logiciel via SSM

  Le domaine utilisé est `/home/ordenv/ssm-domains/ssm-utils`

  Package Linux : `dmv_1.0_linux26-x86-64`

  Package AIX : `dmv_1.0_aix53-ppc-64`

- Compilation C

  Linux : `cc program.c -o program -L/home/ordenv/ssm-domains/ssm-utils/dmv_1.0_linux26-x86-64/lib -ldmv`

  AIX : `xlc -brtl -q64 program.c -o program -L/home/ordenv/ssm-domains/ssm-utils/dmv_1.0_aix53-ppc-64/lib -ldmv`

Environnement utilisateur (2)

- Structure du répertoire $HOME/.dmv :
  `new` : Les nouvelles tâches sont écrites ici.
  `current` : Les tâches en cours d'exécution.
  `delayed` : Les tâches à resoumettre plus tard.
  `aborted` : Les tâches qui sont `delayed` pour plus de 48 heures.
  `jobs` : Les scripts produits à partir des steps. Les listings atterrissent là aussi.

  La commande `dmvprepare` doit être exécutée avant la première utilisation.

- Nom des tâches/jobs :
  `host_pid_time`
  ex : `c6f01p1m_10522_1290103339.103208`

Exemples d'utilisation

- Simple

```
dmvinit_c();
dmvcopy_c("-r -b 4M", "hosta:/directory1", "hostb:/directory2");
dmvlaunch_c();
```

- Moins simple...

```
dmvinit_c();
dmvstep_c("", "echo $DMV_CMD_OUTPUT| mail -s 'dmv error' user@domain");
dmvcopy_c("-r -b 4M", "hosta:/directory1/file*", "hostb:/directory2");
dmvsync_c("-au", "hostb:/directory2", "hostc:/directory3");
dmvstep_c("", "");
dmvcopy_c("", "host4:/file*", "directory3");
dmvwait_c();
dmvexec_c("qsub some_job_command_file");
dmvlaunch_c();
```

Exemples d'utilisation (2)

- Contrôle de flux via le datamover
  - Soit jobA, une job de calcul intensif dans la classe development sur zeta
  - Une tâche datamover (tâcheA) est créée, la job suivante (jobB) dépend des fichiers qu'elle transférera. Les dernières directives de la tâcheA pourrait être :

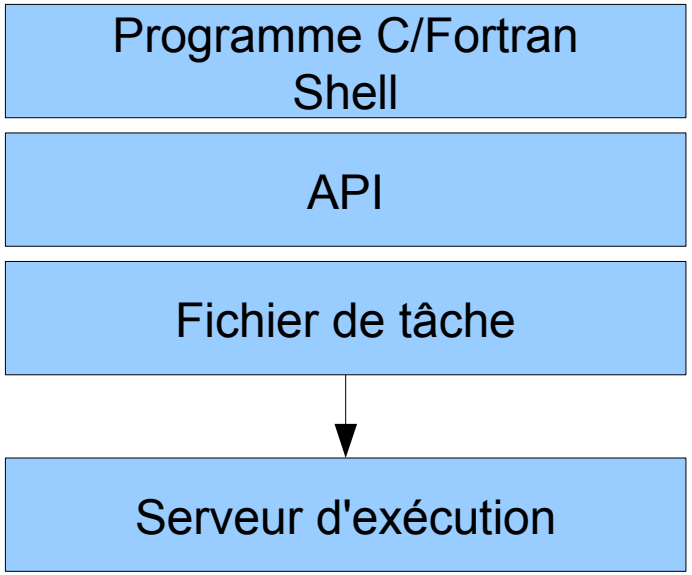
```
dmv_wait_c()
dmv_exec_c("soumet <paramètres pour jobB...>")
```

  - jobA termine
  - tâcheA est traitée et soumet jobB une fois les transferts de fichiers terminés.

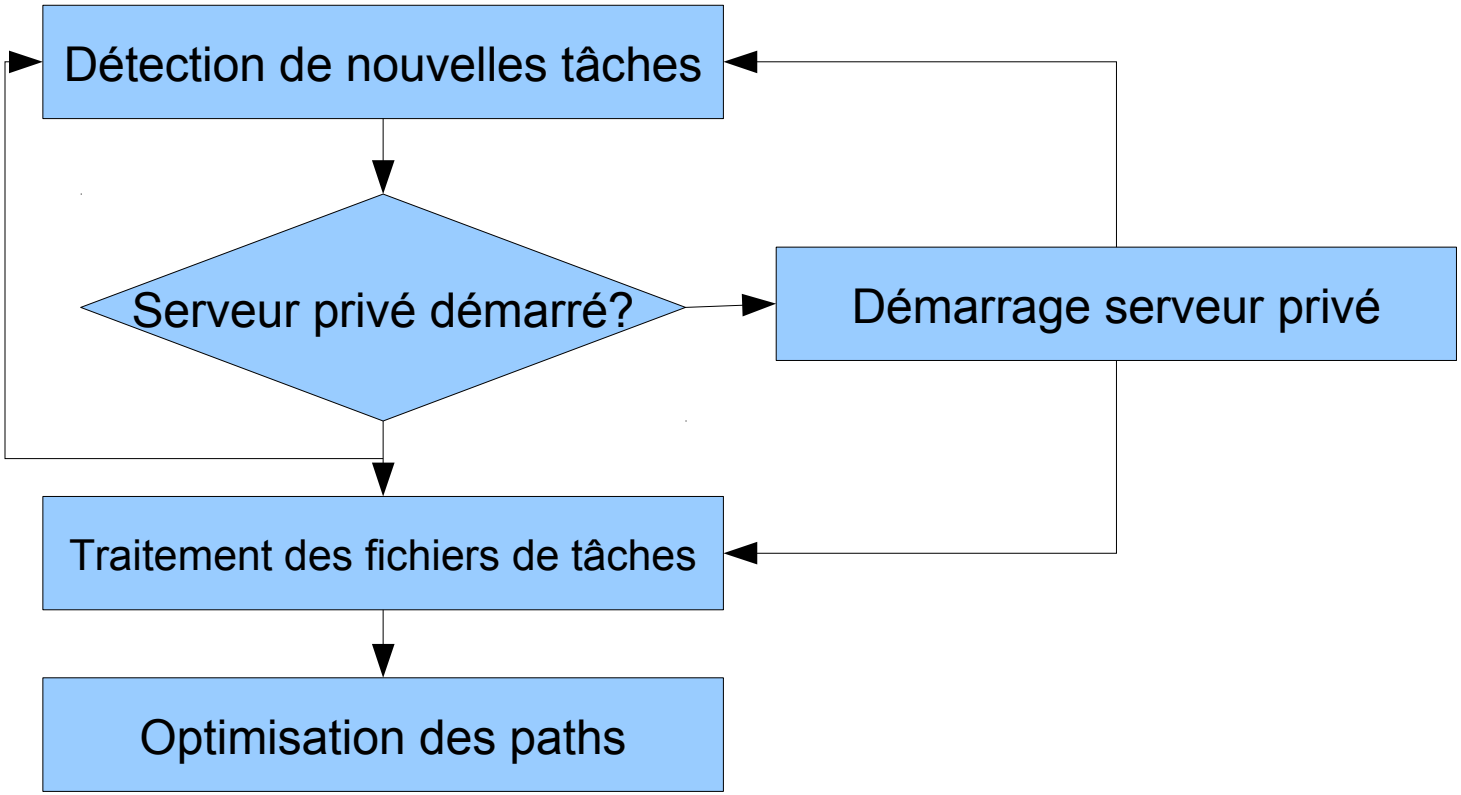
Le serveur d'exécution

- Le serveur d'exécution traite les tâches générées par l'API.
- Chaque *step* d'une *tâche* est converti en *job* qui sera soumise au système batch local (ici, SGE).
- Un processus par utilisateur est utilisé. Aucune commande n'est exécutée avec les privilèges de root.
- Le serveur modifie les chemins source et destination de sorte que les accès se fassent le plus près d'où les fichiers résident et utilisent les réseaux les plus performants.
- Implémenté en python.

Le serveur d'exécution (2)



Serveur d'exécution (3)



Serveur d'exécution (4)

- *Optimisation des paths*

  À partir de l'origine et du répertoire courant à partir desquels la tâche a été créée, les paths cibles (source et destination) sont optimisés en se basant sur l'évaluation des critères suivants :
  - Est-ce que l'hôte cible fait partie d'un cluster?
  - Est-ce que le path cible fait partie d'un clustered file system (par exemple GPFS)?
  - Est-ce que le path cible est accédé via NFS et si oui, quel en est le serveur?
  - Si les deux paths cibles font partie de clustered file systems, sont-ils directement accessibles à partir d'un même cluster?
  - Cas particulier : si un hôte cible n'est pas disponible, la tâche est mise en attente.

Serveur d'exécution (5)

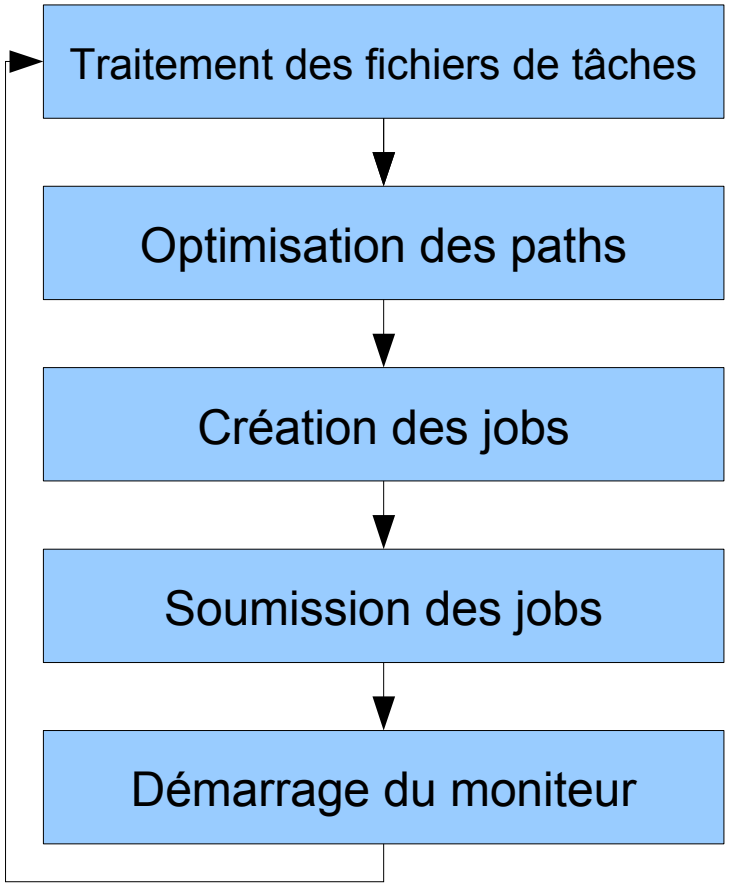
- *Optimisation des paths (suite)*

  Un cluster est défini selon les paramètres suivants :

  - headnodes : les membres d'un cluster qui sont habituellement branchés aux réseaux les plus performants.
  - contact : hôte qui est le point de contact pour l'exécution des commandes de transfert.
  - homeOf : liste des clustered file systems gérés par ce cluster.
  - clusterFS : liste des clustered file systems directement accessibles (inclus homeOf).

  Pour les transferts visant un NAS (metro, bobcat et cie), un hôte cible est prédéfini.

Serveur d'exécution (6)



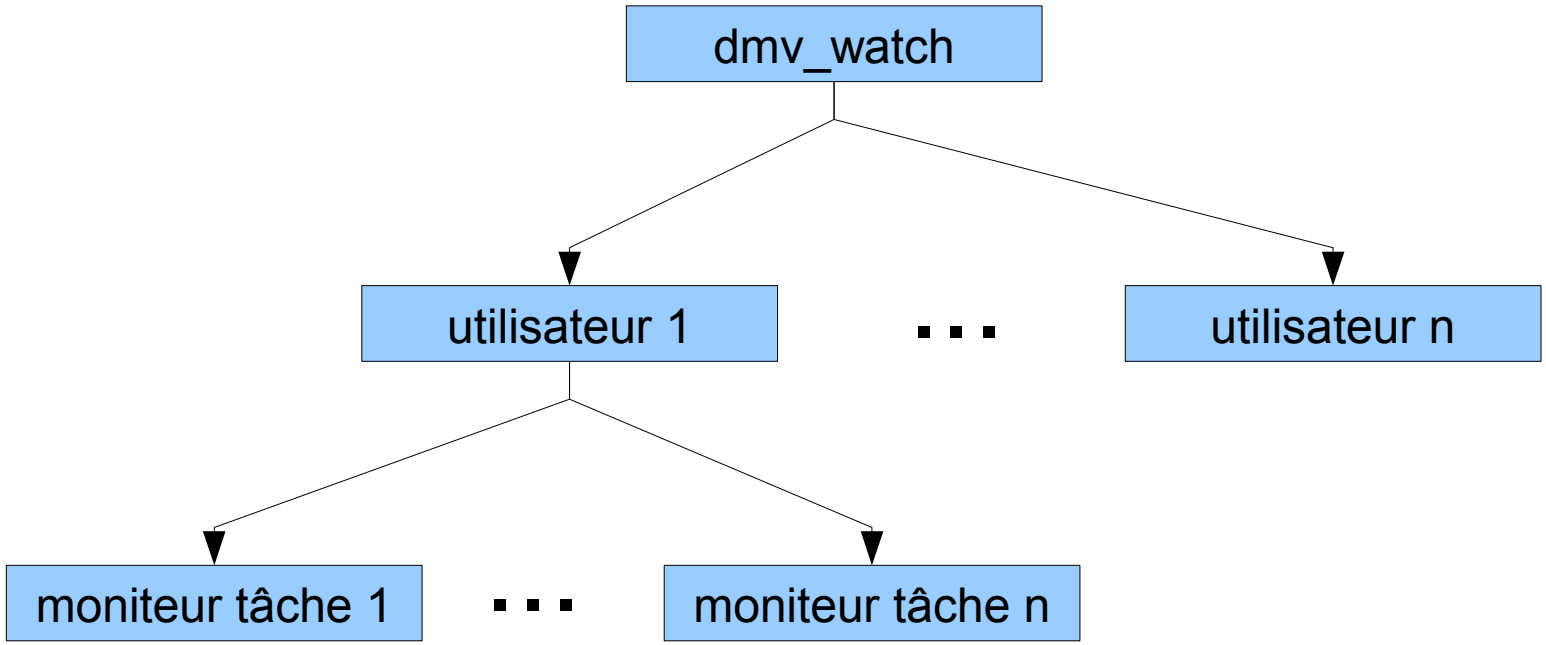
Serveur d'exécution (7)

- Les jobs de transfert :
  - Jobs exécutées par SGE à partir d'un de deux serveurs virtuels (dataq1 et dataq2) configurés en haute disponibilité.
  - Contrôle par « ressource ». Possibilité de retenir l'exécution des jobs pendant les temps systèmes.
  - Chaque commande de transfert peut être essayée un maximum de 3 fois.
  - Présentement, les commandes spécifiées via dmv_exec() ou dmv_step() sont exécutées sur dataq1 ou dataq2

Serveur d'exécution (8)

- Moniteur d'exécution :
  - Pour chaque tâche, un processus est démarré pour surveiller et rapporter via syslog les succès ou erreurs des jobs qui en dérivent.

Serveur d'exécution (9)



Questions?

Quelques réponses sur
https://wiki.cmc.ec.gc.ca/wiki/Datamover