FRANCE GRILLES

Hiérarchie et composition des ressources iRODS

Jérôme Pansanel et Emmanuel Medernach

14 janvier 2021

Crédits

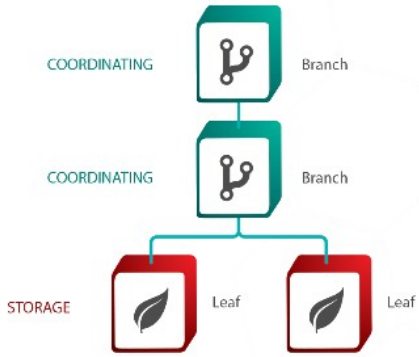
Cette présentation est basée sur la présentation cadre d'iRODS réalisée par Jason Coposky (directeur exécutif, consortium iRODS) :

- https://slides.com/jasoncoposky

Introduction à la hiérarchies des ressources

Hiérarchie des ressources

- Utilise une métaphore de branches et de feuilles (arbre décisionnel)
- Les politiques sont implémentées dans les branches (réplication, cache, ...)
- Deux types de nœuds :
  - Coordination – mise en œuvre pure d'une politique (par ex. passthru)
  - Ressource de stockage – gère l'interface avec la technologie de stockage (par ex. unixfilesystem)
- Par convention, les ressources de coordination ne gère pas de stockage



Plugins pour les ressources

Plugins

- Définit en interne les interfaces vers les technologies de stockage supportées
- Chargé de manière dynamique à l'exécution
- Utilise un système de vote pour informer de la possibilité de satisfaire une opération donnée
- Maintient une configuration individuelle par instance dans le catalogue à l'aide d'une *context string*
- Peut exister indépendamment ou être connecté à d'autres dans une hiérarchie

Motivation

Pourquoi ce système de hiérarchie ?

- Beaucoup d'administrateurs iRODS passent un temps considérable à implémenter des cases d'usage classique avec le même de politiques :
  - Réplication
  - Distribution de données
  - Synchronisation des réplicas
  - Archivage de données
- La hiérarchie des ressources permet de fournir un système prêt à l'emploi pour implémenter la majorité de ces cas d'usage et ce, de manière fiable

Ressources de coordination (branches)

Les ressources de coordination

- **compound** – fournit une interface POSIX vers une composition de deux stockages (par ex. un cache et un stockage classique)
- **deferred** – assemble de une ou plusieurs ressources et a un rôle sur le vote
- **load_balanced** – utilise la charge pour déterminer le choix (uniquement pour l'écriture)
- **passthru** – pondère puis délègue les opérations à une ressource fille
- **random** – choisit une ressource fille aléatoirement pour les opérations d'écriture
- **replication** – s'assure que tous les objets sont constitants entre les ressources filles
- Ces ressources de coordination sont purement virtuelles et en mémoire. Elles ne sont pas liées à un serveur et les plugins associés doivent être déployés sur tous les serveurs de l'infrastructure.

Ressources de stockage (feuilles)

Différents types de stockage

- Les ressources de stockage fournissent une sémantique POSIX
- **unixfilesystem** – n'importent quel type de point de montage
- Ceph-RADOS – Ceph object storage
- HPSS – accès à IBM High Performance Storage System
- Cacheless S3 – ressource pour le service Amazon S3 service
- La ressource est normalement attachée à un serveur identifié (nom d'hôte, chemin)
- **Exception**: la ressource S3 est en mode détaché et n'a pas besoin d'être associée à un serveur
- Ce type de plugins n'a pas besoin d'être installé sur chaque serveur

Ressources pour l'archivage (feuilles)

L'archivage

- La composition avec une ressource compound permet de créer un système de cache
- Par exemple :
  - S3 – ressource archive pour Amazon S3
  - WOS – DDN Web Object Scalar
  - **univmss** – script basé sur un accès générique vers un stockage objet
  - **structfile** – pour s'interfacer avec des fichiers d'archive (zip, tar, ...)
- Doit être lié au serveur hébergeant le cache pour synchroniser les données vers les ressources archives

Mécanisme de vote

La sélection de la ressource de stockage

- Les votes sont par convention entre 0,0 et 1,0
- La communication début à la racine d'une hiérarchie
- Il est interdit de s'adresser aux sous-ensemble directement
- Le vote suit une descente récursive en profondeur
- Les ressources de coordination délèguent le vote aux ressources filles
- Les nœuds de stockage votent
- Les ressources de coordination interprètent les résultats des ressources filles
- L'interprétation de ces votent expriment la politique de gestion de données encodées dans les plugins de coordination

Système de poids pour la ressource *passthru*

Une pratique utile pour le vote

- Pour les opérations *put* et *get* :
- Délégation du vote aux ressources filles
- Multiplication par le poids
- Passe le résultat à la ressource appelante (par ex. rootResc)
- Cela permet de désactiver l'écriture ou la lecture vers une ressource, ou fournir une abstraction aux utilisateurs
- La pondération peut être surchargée par le moteur de règle qui permet une influence dynamique des votes en fonction des politiques

Fonctionnement du vote pour le type *unix file system*

Pour les opérations de type *put*

- Si la ressource est indiquée comme étant *down*, le vote a pour valeur 0,0
- Si le client est connecté au serveur à qui appartient la ressource, le vote a pour valeur 1,0
- Dans les autres case, le vote a pour valeur 0,5

Fonctionnement du vote pour le type *unix file system*

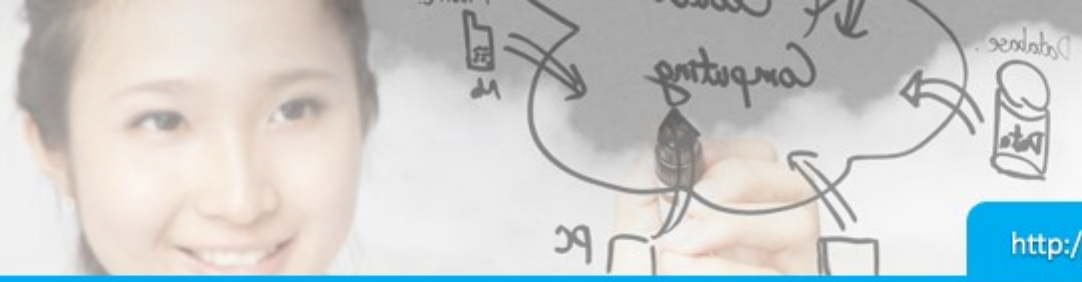
Pour les opérations de type *get*

- Si la ressource est indiquée comme étant *down*, le vote a pour valeur 0,0
- Si le client est connecté au serveur à qui appartient la ressource et que cette ressource a un réplica à jour, le vote a pour valeur 1,0
- Si la ressource a un réplica à jour, le vote a pour valeur 0,5
- Si la ressource a un réplica périmé, le vote a pour valeur 0,25
- Sinon, le vote a pour valeur 0,0

Fonctionnement du vote pour le type *random*

Pour les opérations de type *put*

- Sélectionne aléatoirement une ressource fille et délègue le vote à cette ressource jusqu'à obtenir une valeur de vote positive ou que l'ensemble des ressources aient été questionné
- Passe le résultat à la ressource appelante



Fonctionnement du vote pour le type *random*

Pour les opérations de type *get*

- Délègue le vote à toutes les ressources filles
- Sélectionne la valeur de vote la plus élevée
- Passe le résultat à la ressource appelante

Fonctionnement du vote pour le type *replication*

Pour les opérations du type *put*

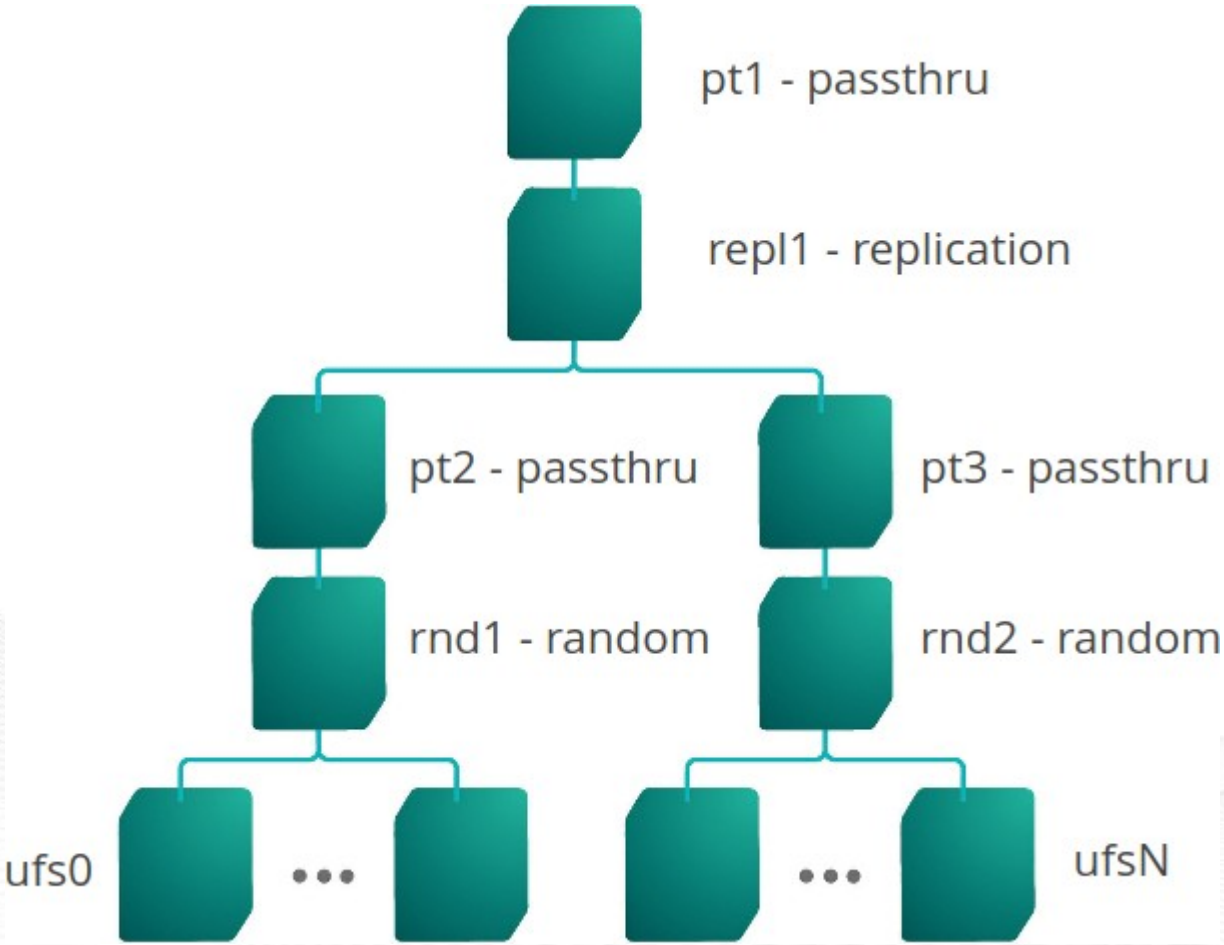
- Délègue le vote à toutes les ressources filles
- Sélectionne la valeur de vote la plus élevée
- Passe le résultat à la ressource appelante
- Une fois que l'action *put* est terminée, lance la réplication vers toutes les autres ressources filles qui acceptent l'opération *put* (par ex. celles qui ne sont pas *down*)

Fonctionnement du vote pour le type *replication*

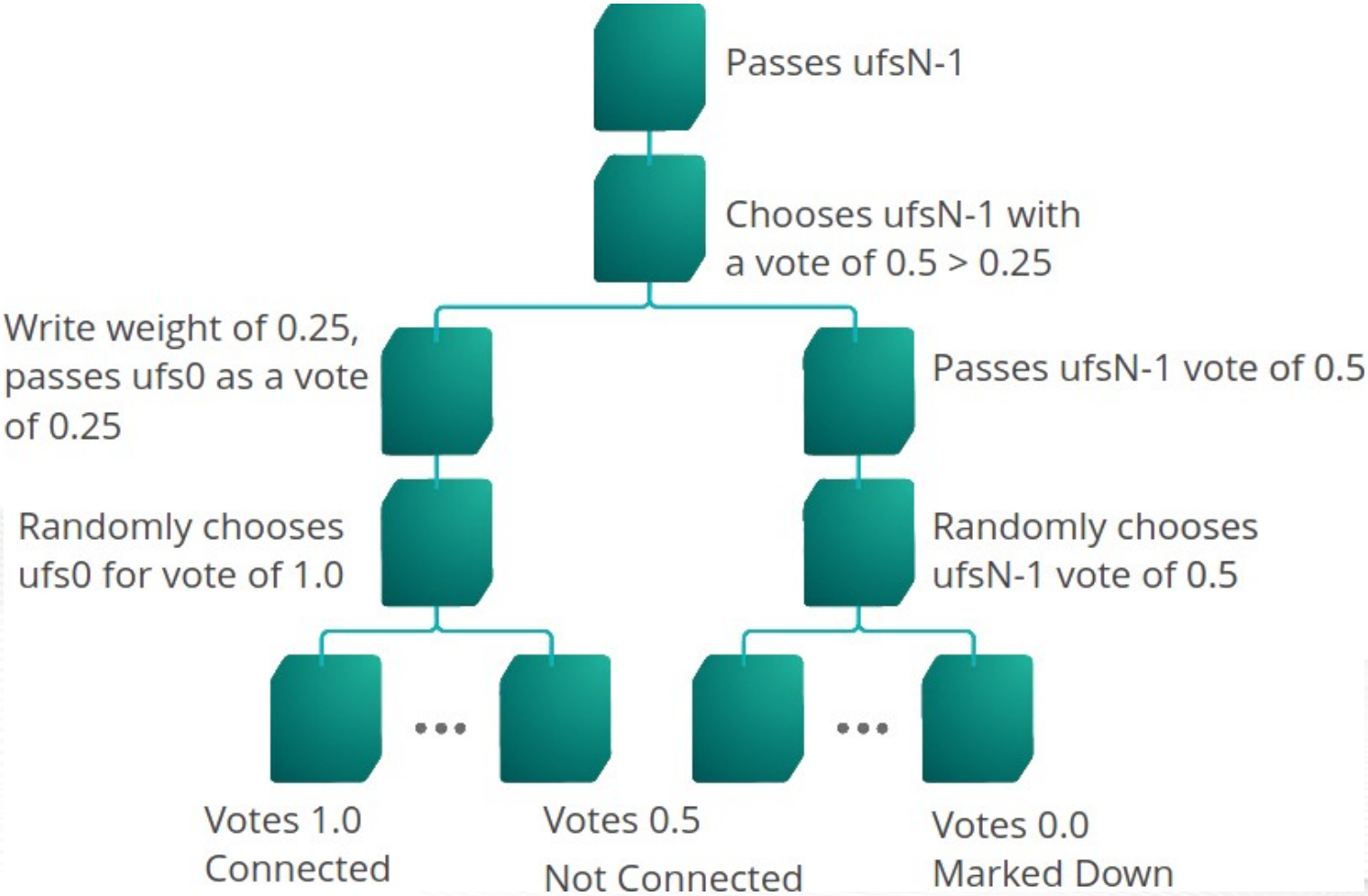
Pour les opérations du type *get*

- Délègue le vote à toutes les ressources filles
- Sélectionne la valeur de vote la plus élevée
- Passe le résultat à la ressource appelante
- Il faut noter qu'étant donné le comportement du système de fichiers (Unix File System), la localité de référence affecte de manière significative le comportement des lectures et des écritures

Fonctionnement du vote : exemple pour une opération *put*



Fonctionnement du vote : exemple pour une opération *put*



Réplication

Réplication entre deux ressources

- Nécessite au moins deux ressources de stockage
- Par défaut, création de réplicas sur toutes les ressources filles
- Utilisation de la chaîne de contexte `num_repl=N` pour contrôler le nombre de réplicas
- Par défault, la réplication est effectuée de manière synchrone
- Si une ressource est down pendant un temps, la commande **rebalance** permet de recopier les fichiers manquant :

```
$ iadmin modresc replResc rebalance
```

Réplication

Détacher la ressource existante

- Suppression de la ressource

```
$ iadmin rmresc resource1
```

- La commande échouera pour au moins une raison : elle est une ressource fille. Il faut donc la détacher :

```
$ iadmin rmchildfromresc rootResc resource1
```

- Il se peut que la commande échoue si la ressource contient encore des données. Il faut les supprimer au préalable :

```
# liste des fichiers dans la ressource resource1
$ iquest "%s/%s" "select COLL_NAME, DATA_NAME WHERE RESC_NAME like
'resource1%'"
# suppression des fichiers existants avec irm -f
...
$ ilsresc
rootResc:passthru
```

Réplication

Créer la ressource replication

- iadmin mkresc nom_de_la_ressource type_de_ressource

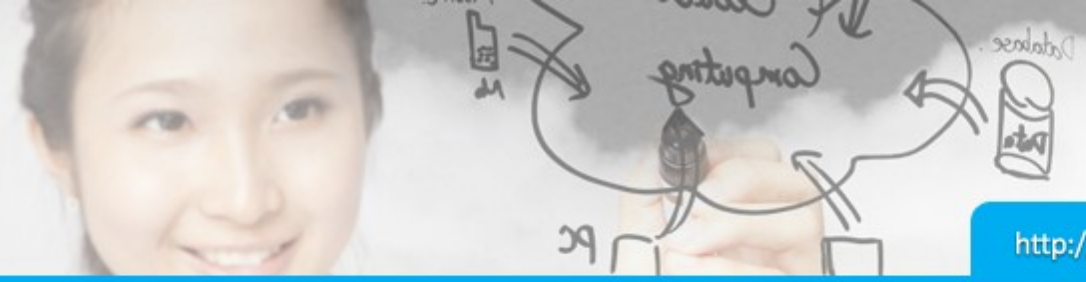
```
$ iadmin mkresc replResc replication
```

- iadmin addchildtoresc parent_name child_name parent_child_context_string

```
$ iadmin addchildtoresc rootResc replResc
```

- Ajout des ressources

```
$ iadmin mkresc resource1 unixfilesystem \
  resource1-X.novalocal:/storage
$ iadmin mkresc resource2 unixfilesystem \
  resource2-X.novalocal:/storage
$ iadmin addchildtoresc replResc resource1
$ iadmin addchildtoresc replResc resource2
$ ilsresc -l
rootResc:passthru
└── replResc:replication
    ├── resource1:unixfilesystem
    └── resource2:unixfilesystem
```



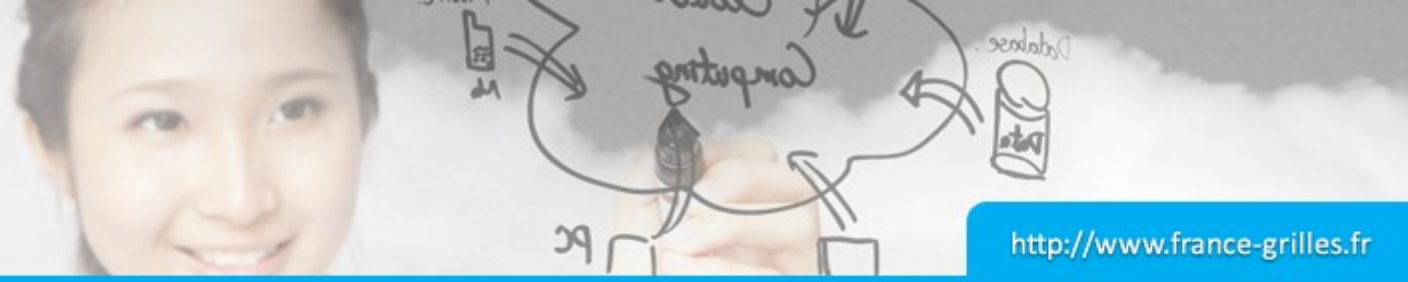
Tests

Des tests pour approfondir

- Copie de fichiers

```
$ iput VERSION.json
```

- Que contient le répertoire `/storage/home/irods` sur chaque serveur de ressources ?
- Éteignez le serveur resource2-X
- Que se passe-t-il si l'un des serveurs est éteint et qu'une copie est effectuée ?
- Comment récupérer des données de rootResc lorsque l'un des serveurs est indisponible ?
- Comment copier vers la ressource rootResc lorsqu'un serveur est indisponible ?
- Que se passe-t-il lorsque le serveur est remis en ligne ?
- Comment retrouver une synchronisation entre les ressources ?



Questions ?