# UE INF4003 : TP 9 et 10
# Mise en œuvre d'une mémoire de tuples distribuée avec Java RMI, ZooKeeper et YARN

*Frédéric Raimbault*

**Remarque :** ce TP est à réaliser en 3 séances consécutives. Il sera testé et évalué à la dernière séance. Le code devra être abondamment et intelligemment commenté.

## Principe de la mise en œuvre distribuée

L'objectif est de réaliser une mémoire de tuples distribuée entre tous les nœuds sur lesquels fonctionne le programme qui l'utilise. Cette mémoire sera liée à l'application sous la forme d'une librairie (i.e. compilée avec l'application) et non pas, accessible comme un service réseau (cf JavaSpaces), partagée entre plusieurs utilisateurs et applications.

La distribution des tuples entre les différents nœuds repose sur deux principes :

1. l'écriture locale : l'utilisation d'une mémoire de tuple locale dans laquelle se fait toute écriture de tuples par un nœud.
2. l'extension des opérations de lecture et de retrait : toute lecture ou retrait d'un tuple commence par une recherche dans la mémoire locale, puis est étendue à l'ensemble des autres mémoires de tuples s'il n'est pas trouvé localement.

Suivant ces deux principes tout tuple est unique, c-à-d. présent dans une seule mémoire locale d'un tuple.

Les tuples considérés seront des listes d'objets et non comme dans JavaSpaces, des objets dont les champs sont les membres du tuple.

L'application qui utilisera la mémoire de tuples prendra en paramètre le nombre de nœuds sur lesquels elle s'exécute et leur code (sous forme d'un jar). Le déploiement de cette application sur un cluster reposera sur l'utilisation de YARN : l'application embarquera un client YARN qui créera un *ApplicationMaster* qui négociera l'exécution des nœuds sur le cluster. A noter que l'*ApplicationMaster* n'aura aucun autre rôle que le déploiement et l'initialisation, il ne servira en aucun cas de serveur de ressources entre les nœuds.

L'accès par un nœud aux mémoires de tuple distantes des autres nœuds sera réalisé à l'aide d'appels de méthodes à distance (RMI). Ces appels nécessitant de connaître la localisation des *servants* des mémoires distantes, les nœuds partageront cette information de groupe à l'aide de ZooKeeper suivant le principe mis en œuvre au TP précédent.

## Travail à réaliser

Vous disposez sur l'ENT :

— de l'implémentation des tuples (`Tuple`),

— de l'interface d'une mémoire de la mémoire de tuples (`TupleMemory`),
— de l'implémentation de la mémoire de tuples locale (`LocalSpace`),
— de la partie générique de l'implémentation de la mémoire de tuple globale (`GlobalSpace`).

1. Commencez par compléter le code de la classe `GlobalSpace` de manière à pouvoir réaliser des appels à distance sur des mémoires de tuples d'autres nœuds.
2. Ajoutez au code précédent la gestion d'un groupe de mémoire de tuple avec ZooKeeper.
3. Écrivez le code de l'*ApplicationMaster* (classe `TupleMemoryApplicationMaster` qui crée l'arborescence Zookeeper nécessaire et négocie avec YARN l'exécution des nœuds sur le *cluster-irisa*.
4. Écrivez le code du client générique d'une application qui utilise une mémoire de tuples (classe `TupleMemoryClient`).
5. Adaptez les exemples de programme *JavaSpaces* vus en cours à cette implémentation distribuée et testez votre mise en œuvre.