Projet ARA : Checkpointing et détection de défaillances

Ce mini-projet a pour objectif de simuler l'exécution d'une application avec PeerSim, et de mettre en place pour cette application un mécanisme de recouvrement sur sauvegarde appuyé par des détecteurs de défaillances.

Etape 1 : Application

On souhaite simuler dans PeerSim une application composée de processus distribués communiquant par messages (NB: vous pouvez reprendre la classe *MatrixTransport* du TME1). On dote chaque nœud d'un compteur « *state* » (*long int*) qui représente l'évolution de son état dans l'application. Chaque nœud incrémente son compteur indépendamment des autres noeuds : le délai jusqu'à la prochaine incrémentation est tiré aléatoirement entre 10 et 20 secondes (NB: vous pouvez utiliser directement *EDSimulator.add* afin de choisir précisément l'heure de réception d'un message envoyé à vous-même). À chaque étape (incrémentation du compteur *state*), on effectue : (i) un tirage aléatoire pour déterminer si l'on envoie ou non un message à un autre nœud choisi aléatoirement (probabilité 0,5) ; et (ii) un pour déterminer si l'on envoie un message à tous les autres nœuds (broadcast, probabilité 0,005).

Testez votre application sur 10 nœuds pendant 3600 secondes (temps de simulation) : faîtes des affichages avec l'id du nœud, l'heure de simulation et la valeur du compteur (i) à chaque incrémentation de compteur *state* ; (ii) à chaque envoi de message ; et (iii) à chaque réception d'un message en provenance d'un autre nœud.

Etape 2 : Checkpointing

Dans cette étape, on vous demande de mettre en place une solution de recouvrement avec des points de reprise non coordonnés suivant l'algorithme de Juang-Venkatesan (JV) vu en cours. Les points de reprise sont sauvegardés indépendamment sur chaque nœud. De la même manière que les changements d'états de l'application (étape 1) le déclenchement de la prochaine sauvegarde peut se faire avec l'envoi d'un message à soi-même (les délais entre les sauvegardes peuvent varier aléatoirement entre 45'' et 1'15''). Chaque nœud doit maintenir des compteurs d'émissions/réceptions de messages pour chacun de ses canaux (i.e., un pour chacun des autres nœuds dans notre application) et un compteur de points de reprise. Ces compteurs, ainsi que l'état de l'application (le compteur *state*), doivent être sauvegardés à chaque checkpoint.

La simulation d'une faute se fait au moyen d'un contrôleur PeerSim qui déclenche un recouvrement au bout d'un délai prédéfini en début de simulation : il contacte un nœud aléatoirement et lui enjoint de revenir à son dernier point de reprise. Ce nœud sera celui qui démarre le recouvrement en suivant l'algorithme de JV. Dès la réception d'un premier message du protocole de recouvrement, chaque nœud cesse d'émettre des messages applicatifs et applique l'algorithme. À vous de déterminer quand un nœud peut reprendre les échanges applicatifs.

Testez votre simulateur en variant (i) la graine du générateur aléatoire (pour avoir des runs différents), (ii) le nombre de nœuds, (iii) vous pouvez également « jouer » avec les probabilités d'envoi de messages et les délais inter checkpoint.

En forçant ou en sautant des sauvegardes d'état et/ou en forçant des émissions de messages, générez un jeu de test qui provoque un effet domino et force le retour de l'application à son point de démarrage.

NB: passer les différentes probabilités, délais, et la graine du générateur de nombres aléatoires via le fichier de configuration permet de conserver aisément un jeu de test.

Etape 3 : Détection de fautes

Dans cette dernière étape, il vous est demandé d'implémenter un détecteur de fautes naïf à base de heartbeats. Le contrôleur injecte des fautes comme dans le TME1. Lorsqu'un nœud suspecte un autre nœud, on simule un détection/redémarrage : il le rallume (*setFailState(Fallible.OK)*) et lui envoie un message de redémarrage. Après réception de messages de détection/redémarrage venant de 3 nœuds différents, le nœud défaillant déclenche le protocole de recouvrement de JV. Un nœud qui défaille en cours de recouvrement cesse de participer au protocole, et attend également 3 messages de détection avant de reprendre sa participation au point où il l'avait interrompue.
Vous devrez également constituer deux jeux de test, cette fois-ci en faisant varier uniquement les points de faute.

Vous rendrez votre projet (code source + fichiers de configuration des différents tests) par e-mail à olivier.marin@upmc.fr et sebastien.monnet@upmc.fr avec dans l'objet :
« ARA – projet – nom1 – nom2 » où *nom1* et *nom2* sont les noms de famille des deux membres du binôme.
Date limite : 1 février 2015 à minuit.