Bureau d'études

Tolérance aux Fautes

1 Contexte et objectifs

1.1 Spécification fonctionnelle

Un service logiciel S acquiert des mesures au travers d'un capteur C. Le service calcule une fonction sur une fenêtre glissante de n valeurs numériques. Le résultat de la fonction de calcul sur les entrées du capteurs est retourné à l'environnement (affichage écran).

1.2 Spécification non-fonctionnelle

Une AMDEC montre que ce service S peut conduire à une défaillance catastrophique du système SYS dans laquelle il est utilisé, en cas d'*erreur en valeur* ou d'*absence de valeur* en sortie. Ce service S doit donc garantir des propriétés de sûreté de fonctionnement, en présence de *fautes transitoires* et de *fautes permanentes*.

1.3 Architecture matérielle

L'architecture matérielle du calculateur qui exécute ce logiciel est un bi-processeur à mémoire commune possédant une mémoire stable sur disque/fichiers (cf. Figure 1). Chaque processeur P_i possède sa propre mémoire locale qui est indépendante. Le capteur ainsi que la mémoire stable (disques) sont accessibles indifféremment pour les deux processeurs. Sur la figure S_1 et S_2 représentent deux copies redondantes du service S.

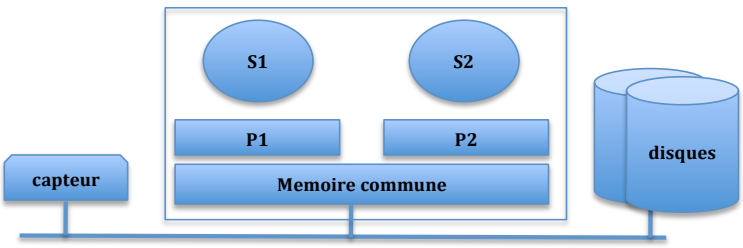

FIGURE 1 – Support matériel

1.4 Objectifs

Nous allons réaliser ce service ainsi que les mécanismes de tolérance aux fautes permettant de tolérer les fautes en crash et en valeur. Plusieurs étapes de travail sont proposées :

— Réalisation du service nominal, sans capacité de tolérance aux fautes.
— Réalisation d'un *Failure detector* (watchdog).

— Ajout d'une capacité de recouvrement par mémoire stable.
— Mise en place du fonctionnement en mode duplex (primary/backup replication) pour la tolérance au crash.
— Mise en place d'un mécanisme de redondance temporelle pour la tolérance aux fautes transitoires en valeur.

Des tests par *injections de fautes* permettront de valider les différents mécanismes réalisés. Un rapport sera demandé à l'issue de ce Bureau d'Etude.

2 Conception

Cette section décrit différents éléments de conception du service et des mécanismes de tolérance aux fautes que l'on souhaite réaliser. Cette description doit permettre de réaliser les différents diagrammes de séquences nécessaires à la conception détaillée.

Ces éléments de conception sont des propositions de solutions permettant d'atteindre les objectifs. Vous avez bien-entendu la possibilité de proposer et d'utiliser d'autres mécanismes.

2.1 Fonctionnement en mode nominal + Failure Detector + Mémoire Stable

Les composants principaux fournissant le service sont représentés sur la figure 2. Le service fournit la fonction à partir des données *capteur*. Il a accès au *Failure Detector* pour déclarer son état *"I'm Alive"* (autrement dit *kick* du *watchdog*). La *mémoire stable* sur disque lui permet de stocker de façon sûre les données capteurs.

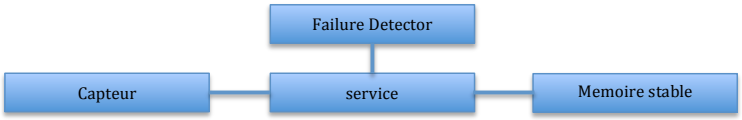

FIGURE 2 – Principaux composants

2.2 Fonctionnement en mode duplex

Le service s'exécute de façon redondante selon une variante du protocole *Primary/Backup*. Le service réalisé fonctionne donc selon deux modes, *Primary* ou *Backup*.

2.2.1 Fonctionnement en mode *PRIMARY*

1. coup de pied au chien de garde
2. Lecture capteur
3. Calcul valeur de sortie des n dernières valeurs
4. Affichage valeur de sortie
5. Stockage infos capteur sur disque (mémoire stable)
6. Retour en 1

Ce fonctionnement en l'absence de faute suit un modèle appelé *BEFORE/PROCEED/AFTER*

— BEFORE : réveil *wactchdog*

— PROCEED : lecture capteur + calcul valeur + affichage
— AFTER : gestion du point de reprise

Les parties BEFORE et AFTER correspondent au protocole non-fonctionnel de gestion de la redondance, la partie PROCEED correspond à l'activation du service fonctionnel. Dans la partie AFTER, le point de reprise (*checkpoint*) dépend cependant de l'état fonctionnel de l'application. Ce modèle permet de distinguer, de façon propre, le fonctionnel du non-fonctionnel (*separation of concerns in english !*).

2.2.2 Fonctionnement en mode BACKUP du service

1. Lecture Watchdog
2. si "Primary Dead" alors
   — Activation procédure de recouvrement
   — Basculement du fonctionnement en mode primaire
3. Sinon retour en 1

Lors du crash, la fonction de recouvrement doit :
— lire les n dernières valeurs dans la mémoire stable (récupération du checkpoint)
— les stocker en mémoire dans le BACKUP

2.3 Capteur

Le capteur est partagé et donc reste accessible au backup en cas de crash du primaire.

Les valeurs délivrées par le capteur seront générées de manière aléatoire. On pourra tester le programme avec une séquence de valeurs connues a priori qui pourront servir d'oracle de test.

2.4 Service

La fonction de calcul sur une fenêtre glissante n'est pas imposée (par exemple, moyenne arithmétique, moyenne pondérée, écart type, distribution de valeur,...) et prend en paramètre n valeurs réelles.

2.5 Failure Detector

Pour réaliser une implémentation redondante du service de calcul, un *failure detector* est nécessaire.

Le mécanisme de détection du crash du service sera réalisé sous forme d'un *watchdog* logiciel. En pratique :
— Le primaire met à jour périodiquement un compteur qu'il met à disposition du secondaire.
— le secondaire surveille l'évolution du compteur de manière périodique. S'il observe que le compteur n'évolue pas, alors il déclare que le primaire est défaillant par crash et déclenche le recouvrement.

2.6 Recouvrement par mémoire stable

En cas de crash le service doit reprendre et calculer la fonction sur une fenêtre glissante des n dernières valeurs.

Le primaire sauve en mémoire stable (un fichier partagé) les valeurs acquises. Lorsque le crash est détecté, le secondaire passe en mode primaire. Il relit les $n-1$ dernières valeurs acquises et stockées par le primaire. Il capture la prochaine valeur au travers du capteur et calcule la fonction sur la fenêtre glissante de n valeurs.

2.7 Composant auto-testable

On veut dans un second temps réaliser un composant auto-testable pour tolérer les fautes aléatoires matérielles transitoires conduisant à des fautes en valeur. Le principe technique proposé est de type *redondance temporelle* (deux exécutions séquentielles de la fonction) avec comparaison des valeurs de sortie. En cas de désaccord, une troisième exécution sera jouée et un vote majoritaire sera utilisé pour départager les résultats. Si le désaccord subsiste, le service s'arrête pour assurer l'hypothèse de *silence sur défaillance*.

On peut considérer que la comparaison est un *test d'acceptation* de la sortie. En reprenant le schéma général d'un composant auto-testable, le contrôleur est ici une version identique de la fonction.

Ce mécanisme sera composé avec le mécanisme de réplication déjà réalisé pour tolérer à la fois les fautes en crash et les fautes en valeur.

3 TO DO !

3.1 Analyse et modélisation du système

On s'appuiera en particulier sur des diagrammes de séquences pour décrire et préciser les différents mode de fonctionnement du système.

3.2 Développement du système

Le développement sera réalisé en plusieurs étapes :
— Réalisation du mode simplex (fonction + failure detector + mémoire stable)
— Réalisation du protocole PBR
— Composition des mécanismes PBR et TR

3.3 Test et mesures

A chaque étape du développement, il faudra vérifier la correction de la conception et de l'implémentation en réalisant :
— des tests par injection de fautes,
— des mesures de performance

3.4 Rédaction du document de synthèse

Le document de synthèse sera à retourner avant le `8 mars 2020`, au format `PDF`, par mail à l'adresse `michael.lauer@laas.fr`. Le plan du document sera le suivant :

1. Introduction
   — Rappel synthétique des objectifs du Bureau d'Etude et de la simulation du système
   — Organisation de votre en groupe et étapes du travail
2. Conception
   — Description de l'organisation conceptuelle de votre logiciel

- Justification des solutions utilisées
- Diagrammes de séquences que vous jugez essentiels

3. Implémentation
   - Donner un descriptif de l'architecture technique de votre réalisation (processus, threads, moyens de communication)
   - Description synthétique de vos réalisation (le code commenté sera à mettre en annexe)
4. Validation
   - Décrire les tests réalisés
     - tests unitaires
     - tests d'intégration effectués
   - Décrire les tests par injection de fautes des mécanismes de tolérance aux fautes
   - Expliquer ce qu'il reste à faire par rapport au problème initial et à sa validation
5. Analyse de la solution et extensions
   - Analyse critique :
     - Quelles sont les hypothèses, les limites et les faiblesses de la solution que vous avez proposé d'un point de vue non-fonctionnel (tolérance aux fautes) ?
     - Discutez l'intérêt du principe de structuration BEFORE/PROCEED/AFTER.
   - Extension :
     - Que doit-on faire si l'on souhaite appliquer vos mécanismes de sûreté de fonctionnement à un autre service applicatif (*réutilisation de vos mécanismes de tolérance aux fautes*) ?
     - Quelles extensions proposeriez-vous en matière de tolérance aux fautes ?
     - Quelles extensions avez-vous déjà réalisé ?
6. Bilan
   - Résumez l'état d'avancement de votre travail
   - Conclure sur l'intérêt, les difficultés rencontrées lors de ce BE, mais aussi sur les limites de l'exercices, et donc sur ce que auriez aimé ajouter ou faire.
7. Annexe
   - Donner le code *commenté* de vos réalisations
   - Donner la documentation permettant d'utiliser votre code (instructions/scripts de compilation, instruction/scripts de lancement, pré-requis, précautions,...)
   - Autre chose d'original que vous voulez exposer