Module 10

Tolérance aux pannes

Michel Dagenais Polytechnique Montréal

Sommaire

1. Terminologie et modèles
2. Réduction et masquage des fautes
3. Réplication
4. Le consensus en réparti
5. Transactions réparties
6. Calcul de disponibilité
7. Cas exemples: les limites de la tolérance aux pannes

Sommaire

1. **Terminologie et modèles**
2. Réduction et masquage des fautes
3. Réplication
4. Le consensus en réparti
5. Transactions réparties
6. Calcul de disponibilité
7. Cas exemples: les limites de la tolérance aux pannes

Terminologie

- Fonctionnement (Dependability)
  - Composantes (Attributes)
    - Disponibilité (Availability)
    - Fiabilité (Reliability)
    - Sécurité (Security)
    - Sûreté (Safety)
    - Confidentialité (Confidentiality)
    - Intégrité (Integrity)
    - Maintenabilité (Maintainability)
- Mécanismes (Means)
  - Prévention des fautes (Fault prevention)
  - Tolérance aux fautes (Fault tolerance)
  - Éradication des fautes (Fault removal)
  - Prévision des fautes (Fault forecasting)
- Problèmes (Impairments)
  - Faute (fault)
  - Erreur (Error)
  - Panne ou défaillance (Failure)

Systèmes fiables

- **Disponibilité** : être prêt à l'utilisation
- **Fiabilité** : continuité de service
- **Sûreté** : pas de conséquences catastrophiques pour l'environnement
- **Sécurité** : prévention d'accès non autorisés et/ou de la manipulation de l'information ((confidentialité, intégrité, disponibilité)
- **Maintenabilité** : réfère à la facilité avec laquelle un système défaillant peut être réparé

- **La fiabilité :** Probabilité pour qu'un système soit continûment en fonctionnement sur une période donnée (entre 0 et t).
- **La disponibilité :** Probabilité pour qu'un système soit en fonctionnement à un instant t donné.
- **La maintenabilité :** Probabilité pour qu'un système en panne à l'instant 0 soit réparé à l'instant t.
- **La sûreté :** Probabilité pour qu'un système soit continûment en fonctionnement non catastrophique sur une période donnée (entre 0 et t)

Quelques définitions (suite)

- **Défaillance (panne) :** survient quand le comportement d'un système viole sa spécification de service
  - Défaillances résultent de problèmes inattendus internes au système qui se manifestent éventuellement dans le comportement externe du système

- Ces problèmes sont appelés **erreurs** et leur causes mécaniques ou algorithmiques sont dites **fautes**

- Quatre sources de fautes peuvent causer la défaillance d'un système :
  - Spécification inadéquate
  - Erreurs de conception dans le logiciel
  - Défaillance du matériel (e.g. processeur, disque, etc.)
  - Réseau de communication

Classification des fautes et défaillances

- **Classification des fautes :**
  - **Transitoire :** se produit de manière isolée (une fois et disparaît)
  - Intermittente **:** se reproduit sporadiquement (transitoire et survient à plusieurs reprises)
  - Permanente **:** persiste indéfiniment (jusqu'à réparation) après son occurrence

- **Type de défaillances :**
  - Plantage ('Crash'): serveur s'arrête, mais il fonctionnait correctement jusqu'à ce qu'il s'arrête
  - **Panne d'omission :** serveur ne répond pas à une requête
  - **Panne de temporisation :** la réponse survient en dehors de l'intervalle de temps réel spécifié
  - Panne de réponse : réponse est simplement incorrecte
  - Panne arbitraire (Byzantine): réponses arbitraires sont produites à des temps arbitraires

Sommaire

1. Terminologie et modèles
2. **Réduction et masquage des fautes**
3. Réplication
4. Le consensus en réparti
5. Transactions réparties
6. Calcul de disponibilité
7. Cas exemples: les limites de la tolérance aux pannes

Approches pour atteindre la Fiabilité

Prévention de fautes : comment prévenir l'occurrence ou l'introduction de fautes

Élimination de fautes : comment réduire la présence (nombre et sévérité) de fautes

Éviter les fautes

Tolérance aux fautes : comment fournir un service en dépit des fautes

Prévision de fautes: comment estimer la présence, la création et les conséquences des fautes

Accepter les fautes

Prévention de Fautes

- **Évitement de fautes** essaie de limiter l'introduction des fautes durant la construction du système par :
  - Utilisation de
    - Composants les plus fiables en respectant les coûts et les contraintes de performances
    - Techniques raffinées d'interconnexion des composants et de l'assemblage des sous-systèmes
    - Langages avec abstraction de données et modularité
    - Environnement de développement et méthodologie structurés aidant à gérer la complexité et ne rien oublier.
    - Méthodologies de conception prouvées
  - Examiner les formes d'interférences
  - Spécification rigoureuse des besoins, si pas formelle

Prévention de Fautes

- Malgré l'évitement de faute, les erreurs de conception (du matériel et du logiciel) existeront
- **Élimination de fautes :** procédures pour trouver et éliminer les causes des erreurs
- Test du système ne peut jamais être exhaustif et éliminer toutes les fautes potentielles
  - Un test peut seulement être utilisé pour montrer la présence de fautes, pas leur absence
  - Il est parfois impossible de tester sous des conditions réelles
  - La plupart des tests sont faits dans un mode de simulation, et il est difficile de garantir que la simulation est exacte
  - Les erreurs introduites durant la spécification des besoins peuvent ne pas se manifester jusqu'à ce que le système tombe en panne

Tolérance aux fautes

- **Récupération de faute**: si un délai de récupération est acceptable.
  - Il doit pouvoir reprendre où on avait laissé, après avoir redémarré (et possiblement réparé) le système.
  - Requiert un stockage permanent pour ne perdre aucune information. Copies de sauvegarde, cliché à intervalle régulier. . .
  - Faire attention de bien tout sauver avant d'accepter une transaction.

Tolérance aux fautes

- Un système est tolérant aux fautes s'il peut masquer la présence de fautes en utilisant la **redondance**
  - **Redondance matérielle** : composants matériels ajoutés pour supporter la tolérance aux fautes
  - **Redondance logicielle** : inclut tous les programmes et instructions utilisés pour tolérer les fautes
  - **Redondance temporelle** : temps extra pour exécuter les tâches (exécuter les instructions plusieurs fois) pour la tolérance aux fautes

Niveaux de tolérance aux fautes

- **Tolérance aux fautes complète :** système continu à fonctionner en présence de fautes avec une perte non significative de fonctionnalités ou de performance
- **Dégradation graduelle (fail soft) :** le système continu à fonctionner en présence de fautes, acceptant une dégradation partielle des fonctionnalités ou de performance durant le recouvrement ou la réparation
- **Arrêt sécuritaire (Fail Safe) :** le système maintient son intégrité tout en acceptant un arrêt temporaire de son fonctionnement

Notes

- Niveau de tolérance aux fautes nécessaire dépend de l'application
- La plupart des systèmes critiques nécessitent une tolérance complète, cependant en pratique plusieurs se contentent d'une dégradation graduelle

Phases de la tolérance aux fautes

- **Détection d'erreurs (DE) :** présence des fautes est déduite en détectant une erreur
  - Aucun schéma de tolérance aux fautes ne peut être utilisé avant de détecter l'erreur associée à ces fautes

- **Recouvrement d'erreurs (RE) :** élimine les erreurs de telle façon qu'elles ne se propagent pas par des actions futures
  - Probablement la plus importante phase de toute technique de TF

Phases de la tolérance aux fautes

- **Recouvrement d'erreurs en aval :** continuer à partir de l'état erroné en faisant des corrections sélectives à l'état du système (Remplacer un disque défectueux dans une unité RAID)

- **Recouvrement d'erreurs en amont (REA) :** consiste à restaurer le système à un état précédent sûr et en exécutant une section alternative du programme
  - **Point de recouvrement :** point auquel un processus est restauré
  - **Point d contrôle (Checkpointing) :** établissement d'un point de recouvrement (en sauvegardant l'état approprié du système)

Traitement de fautes et continuité de service

- Le recouvrement d'erreur retourne le système à un état exempté d'erreurs; l'erreur peut survenir à nouveau. La phase finale de la tolérance aux fautes est d'éradiquer la faute du système
- Une erreur est un symptôme de faute; corriger l'erreur ne répare pas la faute.
- Le composant défaillant (ou la panne) doit être identifié
- Le traitement automatique des pannes est difficile et spécifique au système
- 2 étapes: localisation de la panne et réparation du système
- Techniques de détection d'erreur peuvent aider à retracer la panne
  - Faute matérielle : le composant peut être remplacé
  - Faute logicielle : peut être éliminée dans une nouvelle version du code
  - Dans les applications qui ne doivent pas s'arrêter il est nécessaire de modifier le programme alors qu'il s'exécute!

Masquage hiérarchique des fautes

Un niveau détecte, reprend et masque les erreurs au niveau plus bas:

- Le client demande l'adresse IP d'un ordinateur (DNS).
- Le serveur de nom fait une requête récursive à d'autres serveurs et prend les serveurs alternatifs lorsqu'un serveur ne répond pas.
- Un serveur interrogé découvre une erreur sur un disque et lit plutôt de son second disque en miroir.
- Le client a la réponse et ne sait rien des défaillances.

Masquage par groupe des fautes

- Le client demande à tous les serveurs, prend le premier qui répond. Peut tolérer **n-1** serveurs en panne par omission sur **n** serveurs en total ($\frac{n-1}{n}$) (pannes par omission = serveur ne répond pas à une requête).

- Le client demande à tous les serveurs et prend un vote sur les réponses. Peut tolérer **n-2** serveurs en panne sur **n** serveurs en total ($\frac{n-2}{n}$) si les chances d'avoir deux mauvaises réponses identiques sont presque nulles.

Masquage par groupe des fautes

- Groupe synchronisé: tous les serveurs doivent recevoir les mêmes mises à jour dans le même ordre.

- Groupe non synchronisé: les serveurs de secours enregistrent les mises à jour dans un journal mais sans les traiter. Au besoin, ils utilisent ce journal pour se mettre à jour et prendre le relais avec un certain délai.

Tolérance aux fautes matérielle

- **Statique :** composants redondants utilisés pour cacher les effets des fautes
  - **Exemple :** Triple Modular Redundancy (TMR): 3 composants identiques et un circuit de vote majoritaire; les résultats sont comparés et si un diffère des deux autres il sera masqué
  - Assume que la faute n'est pas commune (tel qu'une erreur de conception) mais elle peut être transitoire ou causée par la détérioration du composant
  - Pour masquer les fautes de plus d'un composant il faut NMR (N-tuple Modular Redundancy)

- **Dynamique :** redondance utilisée à l'intérieur d'un composant indiquant si le résultat est erroné
  - Fournit une technique de détection d'erreurs; recouvrement doit être fait par un autre composant

Tolérance aux fautes logicielle

- Utilisée pour détecter les erreurs de conception
- **Statique :** programmation N-Versions
- **Dynamique :**
  - Détection et recouvrement
  - Blocs de recouvrement : recouvrement d'erreurs en amont
  - Exceptions : recouvrement d'erreurs en aval

Exercice de redondance

Il y a 13 serveurs DNS racine en redondance, qui ont tous le même contenu, pour traduire les noms de domaine en adresses IP. Un client interroge les 13 serveurs pour une requête et retient la réponse qui revient le plus souvent. Combien de serveurs en panne peut-on tolérer avant que ce service DNS racine ne soit plus en mesure de fournir une bonne réponse, si le modèle de panne est

i) détectée (un serveur en panne répond avec un message d'erreur) ?

ii) par omission (un serveur défectueux ne répond pas) ?

iii) panne de réponse (une mauvaise réponse aléatoire) ?

Exercice de redondance - SOLUTION

i. et **ii)** reviennent au même, à la différence que dans le premier cas on sait tout de suite que le serveur est défectueux, alors que dans le second cas on le déduit après un certain délai. On peut tolérer jusqu'à 12 serveurs en panne, car chaque serveur a le même contenu et, même s'il ne reste qu'un seul serveur, il peut offrir le service. Il se peut toutefois que ce serveur devienne très chargé.

iii. on suppose qu'il y a très peu de chances que deux mauvaises réponses aléatoires coïncident, et on peut donc obtenir la bonne réponse s'il reste deux serveurs valides. On peut alors tolérer 11 serveurs en panne.

Sommaire

1. Terminologie et modèles
2. Réduction et masquage des fautes
3. **Réplication**
4. Le consensus en réparti
5. Transactions réparties
6. Calcul de disponibilité
7. Cas exemples: les limites de la tolérance aux pannes

Réplication de données

- **La réplication de données c'est le maintien de copies d'un objet sur plusieurs systèmes.**

- **La réplication permet de :**
  - **améliorer les performances du système selon différents points de vue**
  - **augmenter la disponibilité** en masquant les fautes et donc minimisant l'impact des pannes.
  - Augmenter la performance du système (i.e. 2x plus de bande passante de lecture pour deux disques en miroir, même performance qu'un seul disque en écriture).

- **Performances :**
  - Mise en cache des données accédées fréquemment ou la réplication des données près du point d'accès pour réduire le temps d'accès
  - Répartition des données sur plusieurs serveurs pour répartir la charge de travail
  - La réplication d'objets statiques est triviale tandis que la réplication d'objets dynamique requiert des mécanismes de contrôle de la concurrence et de maintien de la cohérence (ajoute de la surcharge de traitement et de communication)

- **Disponibilité :**
  - Offrir une disponibilité du service le plus près possible de 100%
  - 2 types de pannes :
    - **Serveur** : réplication de données sur des serveurs qui tombent en pannes indépendamment des autres
    - **Partition du réseau** ou opérations déconnectées : les utilisateurs dans leurs déplacements se connectent/déconnectent aléatoirement au réseau

Requis généraux pour la réplication de données

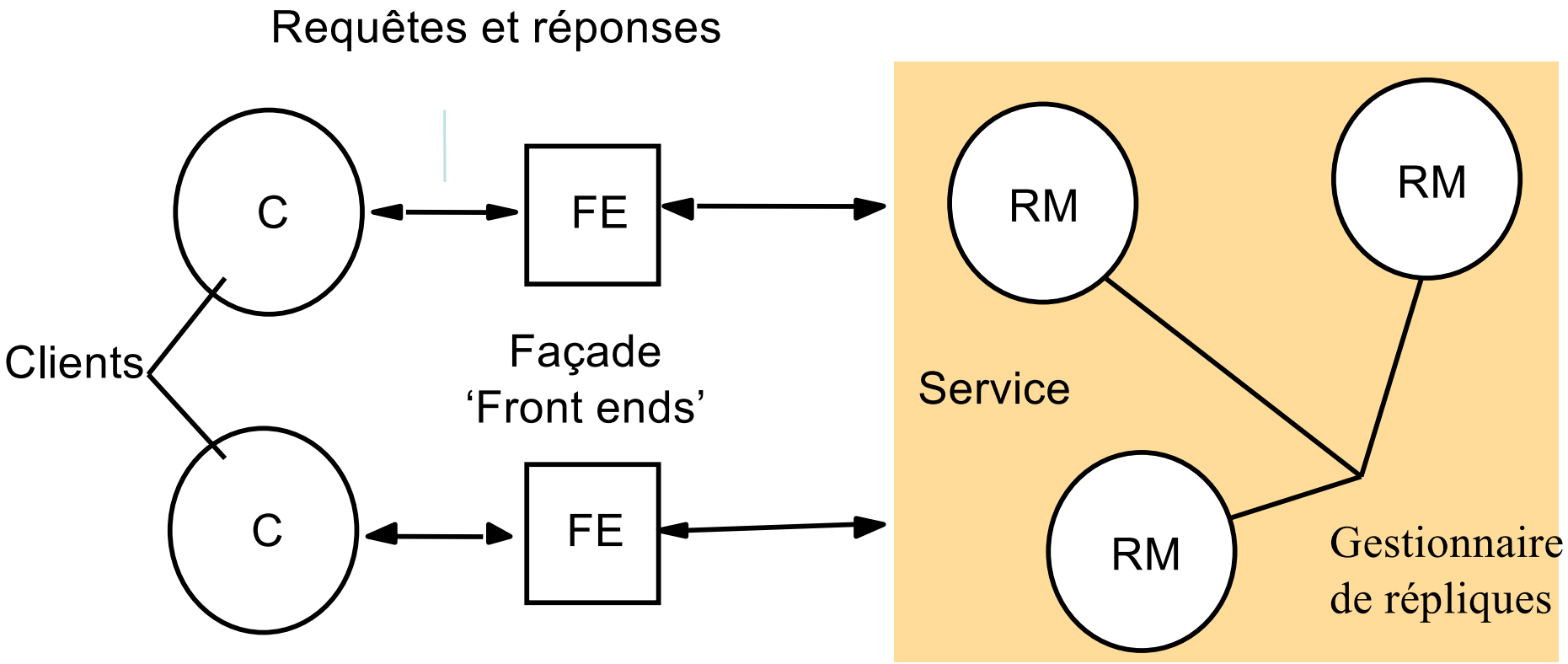
- **Transparence de la réplication :**
  - Les utilisateurs voient des objets logiques, ils n'ont aucune connaissance des multiples copies physiques qui existent
  - Ils accèdent une instance de l'objet, peu importe laquelle, et reçoivent un résultat unique

- **Cohérence :**
  - Étant donné plusieurs copies physiques de l'objet et des accès concurrent, les copies doivent être maintenues dans un état cohérent entre elles

Modèle du système

- Les données dans le système sont considérées comme un ensemble d'objets logiques
- Chaque objet logique est implémenté par un ensemble d'objets physiques appelés répliques
- La cohérence de chacune des répliques dépend des opérations appliquées localement par le système sur lequel elles résident
- Les répliques peuvent être incohérentes entre elles à un point donné du temps (impossible de faire mise-à-jour dans le même instant)
- Le système en soi est asynchrone et peut tomber en panne

Modèle architectural basique pour la réplication



Modèle architectural basique pour la réplication

- Le système se compose de trois éléments :
  - Les clients qui effectuent les requêtes sur des objets répliqués à travers les FE (*Front-End*)
  - Les FE qui agissent comme intermédiaire pour les clients et qui s'occupent de contacter les RMs (replica manager : gestionnaire de répliques). Ils assurent la transparence pour les utilisateurs
  - Les RMs qui maintiennent les répliques d'objets qui sont sous sa responsabilité
    - Les objets peuvent se trouver répliqué sur tout ou sur un sous-ensemble des RMs
    - Les RMs offrent un service d'accès aux objets répliqués au client (accès en lecture ou en mise à jour)

Les étapes de base pour l'accès aux objets répliqués

1) Requête

- Le client effectue une requête sur un objet répliqué
- La requête est prise en charge par le FE qui s'occupent de la relayer à un ou à plusieurs RM par diffusion sélective

2) Coordination

- Les RMs décident d'accepter ou non la requête, lequel d'entre les RMs va desservir la requête et dans quel ordre par rapport aux autres requêtes (i.e. FIFO, causal, total)

3) Exécution

- Le ou les RMs concernés réalisent la requête (peut-être partiellement pour permettre le recouvrement de l'ancien état de l'objet)

4) Accord

- Les RMs se mettent d'accord sur les effets de la requête sur les différentes instances de l'objet répliqué (approche immédiate ou paresseuse)

5) Réponse

- Un ou plusieurs RMs renvoient la réponse au FE ayant effectué la requête, qui lui-même la donne au client concerné

Service tolérant aux fautes

- **Un service a un comportement dit correct s'il fonctionne selon les spécifications en présence de pannes**

- Les clients ne peuvent voir la différence entre un service obtenu à partir de données répliquées et celui obtenu à partir d'une copie unique

- On doit s'assurer que l'ensemble des répliques produit le même comportement qu'une seule copie

Modèle de réplication passive

Dans ce modèle, à tout moment il y a un RM actif (primaire) qui répond aux requêtes des clients et des RMs passifs (secondaires) qui servent de RMs de secours en cas de panne

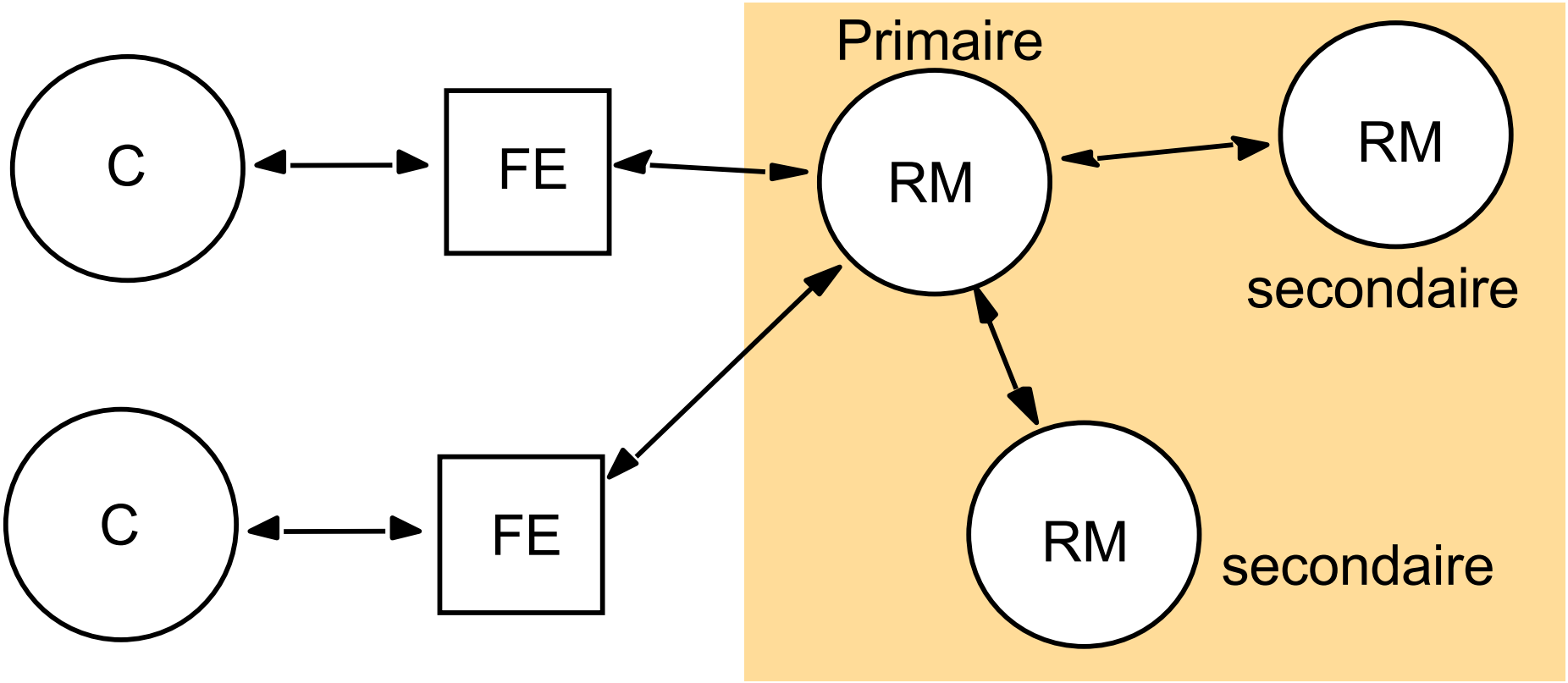
Le résultat d'une opération effectuée sur le RM primaire est communiqué à tous les secondaires (ce qui permet de maintenir la cohérence des données)

Quand le RM primaire tombe en panne, un RM secondaire prend la relève (après une élection, par exemple)

Le système est linéarisable puisque la séquence des opérations est programmée et contrôlée par le RM primaire

Si le RM primaire tombe en panne, le système demeure linéarisable puisqu'un des RMs secondaires reprend exactement là où le RM primaire est tombé en panne.

Modèle de réplication passive



Exemples de réplication passive

- Serveur NIS (Network Information Service) : un serveur primaire qui propage les modifications à plusieurs serveurs secondaires.

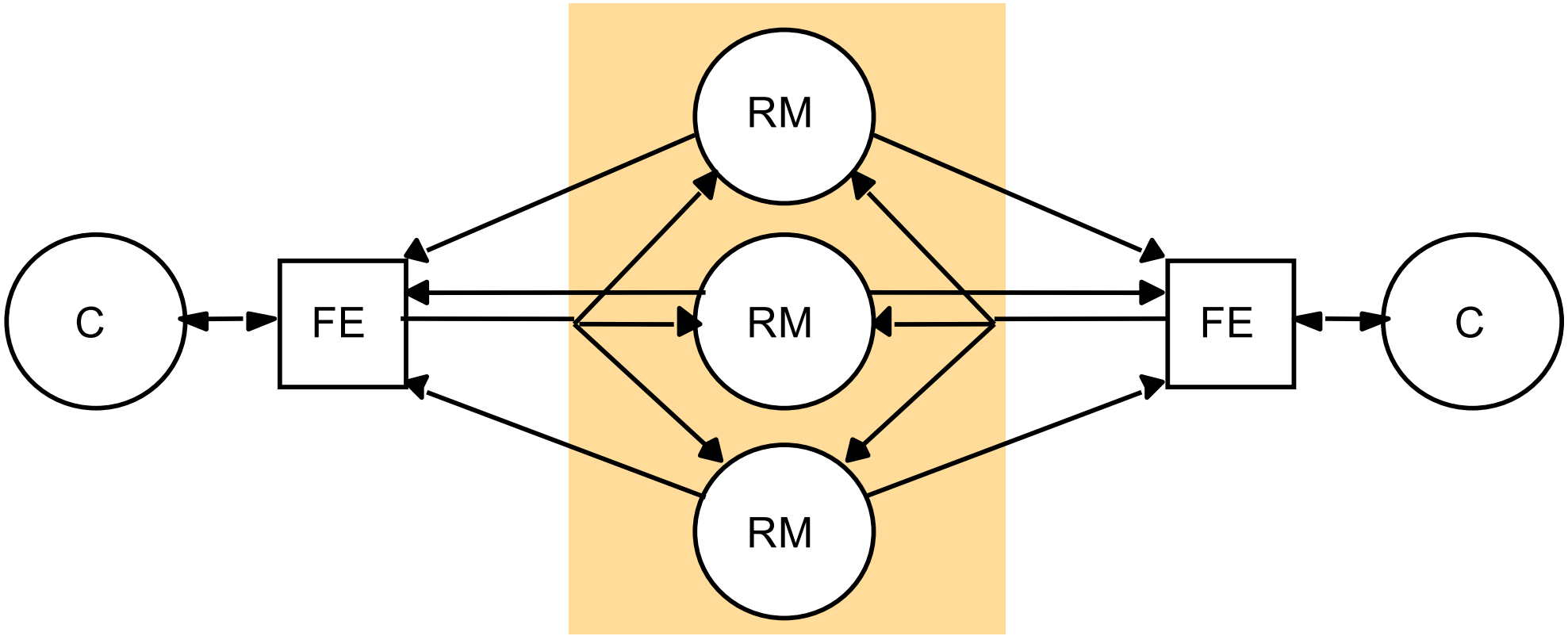
- Linux virtual server: un serveur primaire a deux adresses IP, une pour la gestion et une pour le service. Un serveur de rechange reçoit les mises à jour et interroge souvent le serveur primaire. Si le serveur primaire ne répond plus, il le déconnecte, prend son adresse IP de service et commence à s'annoncer sous cette adresse et à servir les requêtes.

- Deux serveurs DHCP peuvent coexister s'ils offrent des blocs d'adresses différents. Le second serveur répond s'il voit que le premier ne répond pas aux requêtes à tous pour une adresse dynamique.

Modèle de réplication active

- Les RMs débutent dans le même état et changent d'état concurremment de telle manière que leurs états restent identiques
- Si un RM tombe en panne, cela n'a aucun effet sur les performances du système parce que les autres RMs peuvent prendre la relève de manière transparente
- Ce modèle requiert un mécanisme de communication à diffusion sélective (*multicast*) fiable et totalement ordonné (pour que les RMs effectuent les mêmes opérations dans le même ordre)
- Tolère les pannes byzantines (aléatoires)

Modèle de réplication active



Exemple de réplication active

- Serveurs NFS avec ‘automount’: la requête est envoyée à tous les serveurs et le premier à répondre est pris.

- CODA (document editor) : plusieurs serveurs offrent les mêmes fichiers et se propagent les modifications. Très flexible mais ne peut garantir l'absence de conflits (mises à jour concurrentes).

Sommaire

1. Terminologie et modèles
2. Réduction et masquage des fautes
3. Réplication
4. **Le consensus en réparti**
5. Transactions réparties
6. Calcul de disponibilité
7. Cas exemples: les limites de la tolérance aux pannes

Le consensus en réparti

- Plusieurs processus, corrects ou fautifs, échangent des messages.
- Chaque processus doit prendre une décision (e.g. qui est le serveur primaire).
- Terminaison: éventuellement tous les processus corrects arrivent à une décision.
- Consensus: tous les processus corrects terminent avec la même décision.
- Intégrité: si tous les processus corrects proposent la même décision, cette décision doit l'emporter.
- Chaque processus correct communique sa proposition et celle déjà connue d'autres processus au groupe. Chaque processus accumule les propositions des autres et se soumet à la proposition majoritaire.

Difficultés

- Le coordonnateur élu importe peu, **l'important est d'avoir un consensus!**
- L'élection hiérarchique produit un élu dans chaque partition du réseau, ce qui n'est pas acceptable si on doit avoir un coordonnateur unique (verrou, base de données).
- Exiger un vote à majorité? Ceci assure qu'un seul coordonnateur peut être élu.
- Chaque candidat va chercher des votes. Que faire en cas de résultat minoritaire? Peut-on changer son vote?

Algorithme de Paxos (Lamport)

- Le consensus en réparti est très difficile à assurer en présence de défaillances, messages asynchrones, partitionnement de réseau...

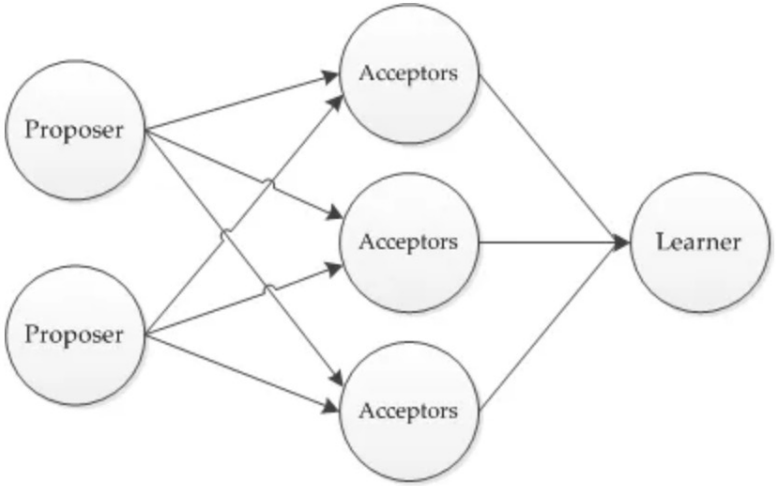
- Complexe, utilisé surtout pour les questions fondamentales (e.g., élire un serveur primaire de manière sécuritaire, ensuite le serveur primaire peut coordonner le reste).

Algorithme de Paxos

- Par exemple, si on veut élire (établir un consensus) un serveur primaire, il faut obtenir une majorité parmi les accepteurs.
- Chaque proposition a un numéro de séquence et une valeur.
- Première proposition reçue par un accepteur, il promet de ne pas accepter une proposition antérieure et mémorise cette dernière proposition.
- Proposition antérieures reçues par un accepteur → retourne la plus récente proposition reçue.
- Le proposeur choisit la proposition la plus récente déjà reçue par un accepteur et demande à chaque accepteur de l'accepter.
- Si une majorité d'accepteurs acceptent, le consensus est obtenu.

Algorithme de Paxos

- Un nœud Paxos peut assumer l'un ou l'ensemble des trois rôles : proposant (proposer), accepteur (acceptor) et apprenant (learner). Un proposant propose une valeur sur laquelle il souhaite obtenir un accord. Pour ce faire, il envoie une proposition contenant une valeur à l'ensemble de tous les accepteurs, qui décident d'accepter ou non la valeur. Chaque accepteur choisit une valeur de manière indépendante — il peut recevoir plusieurs propositions, chacune d'un proposant différent — et envoie sa décision aux apprenants, qui déterminent si une valeur a été acceptée.



Exercice de Paxos

On veut appliquer l'algorithme du consensus de Paxos à l'élection d'un serveur primaire. Ce serveur primaire doit être unique en tout temps afin d'assurer la cohérence du système (e.g. attribuer les sièges d'une salle de spectacle à des clients qui se connectent via un réseau externe). Si le serveur primaire issu du consensus devient éventuellement non disponible, peut-on changer le consensus pour un nouveau serveur primaire? Comment empêcher un serveur primaire coupé du réseau de continuer à se considérer primaire (et vendre des sièges possiblement en double)? Donnez un exemple.

Exercice de Paxos - SOLUTION

Un consensus ne devrait normalement pas changer, le serveur primaire peut cependant devoir changer au fil du temps. Par exemple, on peut avoir un consensus sur le serveur primaire pour un intervalle donné. L'intervalle peut être dénoté par un rang (e.g. 14ème intervalle) ou par un temps de début. En cas de panne de ce serveur primaire, un nouveau consensus peut s'établir pour un nouveau serveur primaire pour le nouvel intervalle (e.g. 15ème intervalle) et tout le monde communique toujours avec le serveur primaire du plus récent intervalle. Le serveur primaire pourrait ne pas être en panne mais simplement coupé du réseau. On ne peut distinguer l'un de l'autre. Il peut alors être justifié d'élire un nouveau serveur primaire si l'ancien n'est pas rejoignable. Si l'ancien serveur primaire est en panne, il ne fait plus rien et ne cause pas de conflit.

Exercice de Paxos – SOLUTION (suite)

Si l'ancien serveur primaire est simplement coupé du réseau et continue à se considérer comme primaire, ce peut être problématique puisqu'il continue `a se considérer en autorité pour effectuer des opérations critiques (attribution des sièges). Une solution que l'on retrouve souvent dans les configurations redondantes à deux serveurs (actif-passif) est que le nouveau serveur commande que l'ancien serveur soit réinitialisé pour prévenir ce genre de problème.

Exercice de Paxos – SOLUTION (suite)

Supposons 5 serveurs A, B, C, D, E. Au moment n=1, A se propose (v=A) comme serveur primaire et obtient une promesse de A, B et C, ce qui constitue un quorum. Presqu'en même temps, au temps n=2, E se propose (v=E) comme serveur primaire et obtient une promesse de D et E. Si A poursuit en demandant à D et E, il peut se faire ignorer ou se faire répondre qu'une proposition plus récente circule. En apprenant qu'une proposition plus récente circule, il peut abandonner. Autrement, il pensera que D et E sont en panne et, ayant déjà le quorum, A envoie un message demandant d'accepter n=1, v=A. Si ceci est effectué très rapidement, et que A reçoit la confirmation de A, B et C, la majorité est obtenue et le consensus atteint. Dans ce cas, E ne réussira pas à avoir un quorum de promesses et encore moins pour son acceptation.

Algorithme Raft

- Reliable/Replicated/Redundant And Fault-Tolerant : RAFT.
- Proposé par Diego Ongaro et John Ousterhout en 2013: "In Search of an Understandable Consensus Algorithm".
- Preuve formelle de sécurité.
- Utilisé par Etcd, MongoDB et plusieurs autres systèmes.

Algorithme Raft

- Lors de son initialisation ou lorsqu'il perd le contact avec le primaire, un membre incrémente le numéro de terme (époque) et se propose comme candidat. Il envoie un message à tous les autres membres pour avoir leur appui.
- Chaque membre ne vote qu'une seule fois pour un terme.
- Si un numéro de terme plus élevé apparaît, l'élection en cours devient caduque et est perdue.
- Si un candidat obtient une majorité absolue de votes, il est élu.
- Les délais avant de demander un vote sont choisis aléatoirement pour minimiser les chances de collisions.
- Le nouveau primaire élu reste en poste aussi longtemps qu'il est opérationnel et que le contact n'est pas perdu.

Le consensus avec Raft

- Le primaire est responsable de coordonner un état répliqué entre tous les membres.
- Toutes les nouvelles entrées sont envoyées au primaire.
- Le primaire met l'entrée dans son journal et l'envoie aux membres.
- Chaque membre ajoute l'entrée dans son journal et répond que c'est accepté.
- Si une majorité de membres acceptent l'entrée, elle est commise par le primaire.
- Le primaire envoie un message pour confirmer que l'entrée est commise.
- Si un nouveau primaire est élu, il vérifie avec chaque membre les dernières entrées commises et les propage à tous.

Sommaire

1. Terminologie et modèles
2. Réduction et masquage des fautes
3. Réplication
4. Le consensus en réparti
5. **Transactions réparties**
6. Calcul de disponibilité
7. Cas exemples: les limites de la tolérance aux pannes

Transactions réparties avec réplication

- Le protocole à deux phases pour compléter une transaction doit s'étendre aux données répliquées qui doivent elles aussi changer de manière atomique avec le reste.

- Le coordonnateur de transaction parle au gestionnaire de copies qui agit à son niveau comme coordonnateur.
  - Les messages 'prêt', et dans une seconde phase 'compléter' ou 'annuler', sont relayés à toutes les copies avant de produire une réponse au coordonnateur.
  - C'est un peu comme si le coordonnateur de transaction devait contacter toutes les copies.

Transactions réparties avec réplication (suite)

- Lecture sur un, écriture sur tous: verrou en lecture sur un, ou verrou en écriture sur tous (bloqué si un verrou en lecture existe).
- Copie primaire: tous les verrous sont pris sur la copie primaire, au moment d'accepter la transaction, toutes les mises à jour sont propagées aux copies.
- Il faut faire attention si les copies disponibles pendant la transaction changent, que le tout demeure cohérent.
- Lorsque le réseau est partitionné, on peut continuer dans chaque partition et résoudre les conflits plus tard (optimiste), ou ne continuer que dans la partition qui a le quorum (si une partition a plus que la moitié et est donc la seule ainsi), ou en être réduit à des lectures seulement puisqu'il n'est plus possible d'obtenir des verrous d’écriture sur tous.

Sommaire

1. Terminologie et modèles
2. Réduction et masquage des fautes
3. Réplication
4. Le consensus en réparti
5. Transactions réparties
6. **Calcul de disponibilité**
7. Cas exemples: les limites de la tolérance aux pannes

Calcul de disponibilité

- Probabilité qu'un composant soit fonctionnel au temps t: $\frac{MTTF}{MTTF+MTTR}$ (Mean Time to Fault / Repair).
- Probabilité p que deux événements indépendants, de probabilité p_1 et p_2, se produisent en même temps: $p = p_1 \times p_2$.
- Probabilité p que p_1 ou p_2 se produise: $p = p_1 + p_2 - p_1 \times p_2$
- Les probabilités de cas disjoints ($p_1 \times p_2 = 0$) peuvent s'additionner.
- Probabilité que k disques sur n soient fonctionnels ($n - k$ en panne), si la propabilité pour un disque est p_d.

$$p = \binom{n}{k} \times p_d^k \times (1 - p_d)^{n-k} \qquad \binom{n}{k} = C_n^k = \frac{n!}{k!(n-k)!}.$$

  - Sommation de k à n pour k disques ou plus fonctionnels.
  - Au moins 1 disque fonctionnel dans un miroir: $1 - (1 - p_d)^2$

Probabilités : Peut-on sommer

- Probabilité que deux événements indépendants de probabilité p1 et p2 se produisent au même moment: $p1 \times p2$. Par exemple, un ordinateur est fonctionnel si la carte mère (pm) et le disque (pd) sont fonctionnels. Il y a une panne sauf si les deux sont fonctionnels.

$$1 - p_m \times p_d$$

- Peut-on plutôt prendre la probabilité de panne de carte + la probabilité de panne de disque?

$$(1 - p_m) + (1 - p_d) \neq 1 - p_m \times p_d$$

Probabilités : sommer si disjoint

- Quelle est la différence? La panne de disque se produit indépendamment de la carte qui peut être fonctionnelle ou non. On compte donc deux fois le cas où le disque et la carte sont défectueux. Les deux éléments sommés se recoupent alors qu'il faut bien tout séparer. Si p1 et p2 sont grands, la partie sommée en double est petite et le résultat est incorrect mais très proche. Si p1 et p2 sont petits, le résultat dépasse largement 1!

$$P(p_1 \cup p_2) = p_1 + p_2 - p_1 \times p2 =$$

$$(1 - p_m) + (1 - p_d) - ((1 - p_m) \times (1 - p_d)) =$$

$$(1 - p_m) + (1 - p_d) - (1 - p_d - p_m + p_m \times p_d) = 1 - p_m \times p_d$$

Systèmes parfaitement tolérants aux pannes?

- Avec le bon niveau de redondance, on peut diminuer les probabilités de panne pratiquement à 0, sous certaines hypothèses.

- Cinq 9 (0.99999) donne 5.26 minutes par an de panne, six 9 (0.999999) donne 31.5 secondes par an.

- En pratique, les pannes sont plus fréquentes car certaines hypothèses ne tiennent pas, surtout sur les nouveaux systèmes.

- Pour les systèmes plus matures, on analyse les accidents et on apprend des erreurs (e.g., en aviation).

Disques en miroir

- Un disque tombe en panne environ une heure par 4 ans (1h/4ans = 0.00003). Avec 2 disques en miroir les probabilités de pannes simultanées sont à peu près nulles ($.00003^2$ = 1h/150000ans). Plus besoin de copies de sauvegarde?!?

- Lors d'une panne, il faut être alerté et effectuer la réparation rapidement pour revenir à une configuration redondante.

- Le matériel redondant doit être en bon état de marche.

- En fin de vie, les probabilités de panne augmentent rapidement. Deux vieux disques deviennent problématiques.

- Il y a parfois des lots de disques non fiables avec fin de vie prématurée. Plusieurs vont acheter des disques de lots / modèles différents.

- Vol, feu, foudre, surtension.

Exercice de disponibilité

Trois ordinateurs offrent un service. Chaque ordinateur fait en moyenne une panne après 5 jours d'opération et cette panne prend 4 heures à réparer. Quelle est la disponibilité de ce système?

La probabilité de ne pas être fonctionnel est de:

$$\frac{4\text{h}}{5 \times 24\text{h} + 4\text{h}} = 0.03$$

La probabilité d'avoir les trois ordinateurs simultanément en panne est de:

$$(0.03)^3 = 0.00027$$

La disponibilité est donc de $1 - 0.000027 = 0.999973$.

Exercice de disponibilité

On vous demande de choisir entre deux configurations pour le prochain serveur de fichiers de votre entreprise. La première configuration consiste en un ordinateur avec 4 disques en miroir. Chaque disque contient l'ensemble des données et un seul disque suffit donc. Le second système est constitué de deux ordinateurs, qui sont deux serveurs redondants, chacun étant connecté à deux disques en miroir et un seul disque suffit donc pour un serveur. La probabilité qu'un ordinateur soit opérationnel (hormis les disques) est de 0.95. La probabilité qu'un disque soit opérationnel est de 0.85. Quelle est la probabilité que le service soit disponible, pour chacune des deux configurations?

Exercice de disponibilité - SOLUTION

- Première configuration:
  - Le système de disque est disponible sauf si les 4 sont indisponibles: $1 - (1 - .85)^4 = 0.99949375$.
  - Le service sera disponible si les disques et l'ordinateur le sont: $0.99949375 \times 0.95 = 0.949519063$.
- Seconde configuration:
  - Sur un serveur, les disques seront disponibles sauf si les 2 sont indisponibles: $1 - (1 - .85)^2 = 0.9775$.
  - Un serveur sera disponible si l'ordinateur et les disques le sont: $0.9775 \times 0.95 = 0.928625$.
  - Le service sera disponible sauf si les deux serveurs redondants sont indisponibles: $1 - (1 - 0.928625)^2 = 0.994905609$.

La deuxième configuration est donc nettement mieux pour réduire le temps d'indisponibilité. Elle est toutefois légèrement plus coûteuse puisqu'elle utilise deux ordinateurs plutôt qu'un, tout en conservant le même nombre de disques.

Exercice de disponibilité

Un client utilise le service DNS et peut contacter 4 serveurs différents. Chaque serveur a une probabilité d'être disponible de 0.6.

i) Si ces serveurs présentent des pannes par omission, combien de serveurs en panne peut-on tolérer et quelle est la probabilité que le service soit disponible au client?

ii) Si les pannes sont de type réponse aléatoire et le client prend un vote parmi les réponses des 4 serveurs, combien de serveurs en panne peut-on tolérer, et que devient la disponibilité du service?

Exercice de disponibilité - SOLUTION

i) Avec des pannes par omission, on peut tolérer jusqu'à 3 serveurs en panne.

- La probabilité qu'un serveur soit en panne = 1 – 0.6 = 0.4
- **La probabilité que les 4 serveurs** soient en panne est de $0.4^4 = 0.0256$.
- La probabilité de disponibilité est = 1 − 0.0256 = 0.9744.

Exercice de disponibilité - SOLUTION

ii) Avec des pannes de réponse aléatoire, il faut avoir **au moins** 2 serveurs disponibles.

Nous avons une probabilité

- de $0.6^4 = 0.1296$ d'avoir les 4 serveurs disponibles,
- de $4 \times 0.6^3 \times (1 - 0.6)^1 = 0.3456$ d'avoir exactement 3 serveurs disponibles parmi les 4,
- et de $4!/(2!(4 - 2)!) \times 0.6^2 \times (1 - 0.6)^2 = 0.3456$ d'avoir exactement 2 serveurs fonctionnels.

Le total est de $0.1296 + 0.3456 + 0.3456 = 0.8208$.

Exercice de disponibilité - Disques

Une compagnie, qui offre des services infonuagiques, met gratuitement à la disposition des usagers un service de fichiers (semblable à Google drive). Cette compagnie opère 30 000 serveurs, constitués en 10 000 groupes de 3 serveurs en redondance. En effet, les fichiers de chaque usager sont dupliqués sur 3 serveurs différents, les 3 serveurs d'un groupe, pour assurer une bonne tolérance aux pannes. Chaque serveur contient une unité RAID de 3 disques. Pour chaque serveur, la probabilité de panne, hormis les disques, est de 0.2. Pour chaque disque, la probabilité de panne est de 0.3, et l'unité RAID peut tolérer un disque en panne sur les 3.

i) Quelle est la probabilité qu'un serveur donné soit disponible?

ii) Quelle est la probabilité, pour un usager donné, que ses fichiers soient disponibles?

iii) Quelle est l'équation (par exemple en fonction de la réponse à i) ou ii) qui donne la probabilité que le service soit entièrement disponible pour tous les usagers?

Exercice de disponibilité – Disques - SOLUTION

i. L'unité de disques RAID a une probabilité de fonctionner de
$0.7^3 = 0.343$ (3 disques fonctionnels)
$3!/((3-2)!2!) \times 0.7^2 \times (1-0.7)^{3-2} = 0.441$ (exactement 2 disques fonctionnels),
pour un total de 0.343 + 0.441 = 0.784. (Unité RAID)
Un serveur est opérationnel si l'unité RAID fonctionne et le reste aussi, ce qui donne **0.8 × 0.784 = 0.6272**.

ii. Le service sera disponible pour un utilisateur sauf si les 3 serveurs du groupe qui le sert sont en panne, une probabilité de $1 - (1 - 0.6272)^3 = 0.948188316$ que le groupe de l'usager soit fonctionnel.

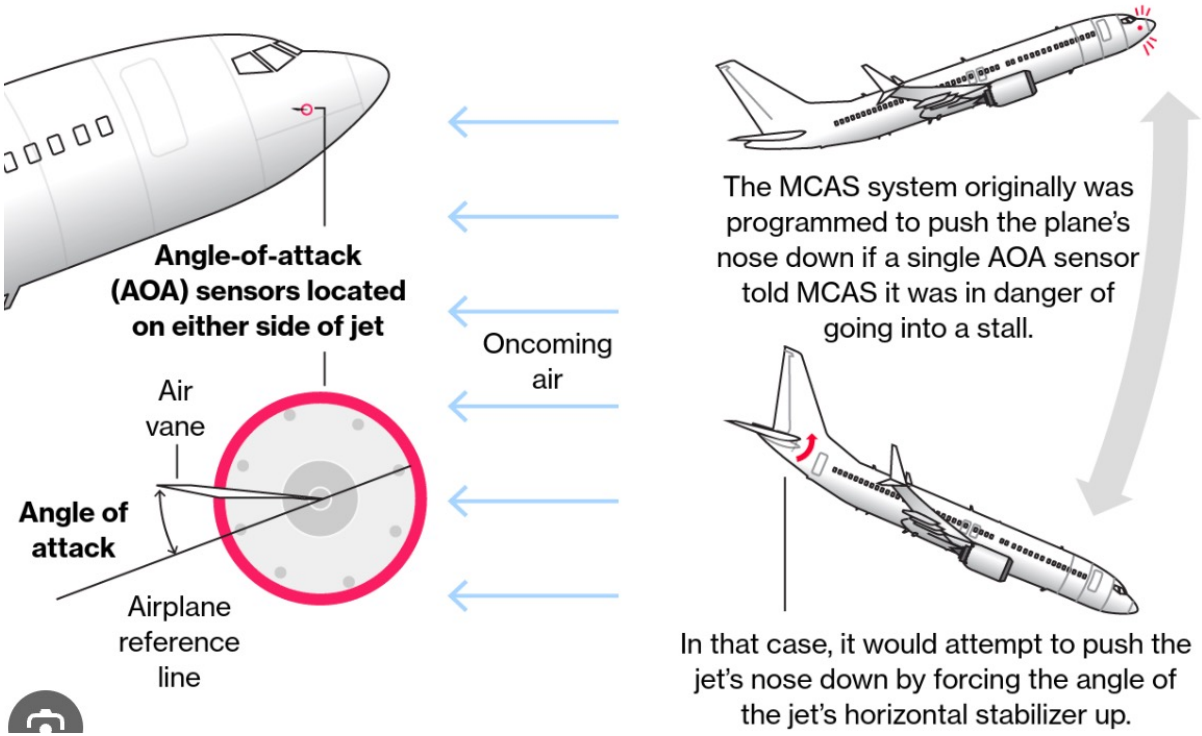
iii. Le service sera disponible pour tous les usagers si tous les groupes de 3 serveurs sont fonctionnels pour les 10000 groupes, une probabilité de 0.948188316^{10000} qui est pratiquement nulle

Sommaire

1. Terminologie et modèles
2. Réduction et masquage des fautes
3. Réplication
4. Le consensus en réparti
5. Transactions réparties
6. Calcul de disponibilité
7. **Cas exemples: les limites de la tolérance aux pannes**

Accident BOING737-MAX

Les failles dans la conception du logiciel qui ont retiré aux pilotes le contrôle du vol à leur insu, sur les données d'un seul capteur, ont finalement conduit aux deux crashs du 737 MAX en 2018 et 2019, causant la mort de 346 personnes.



Accident BOING737-MAX

- Lors du crash d'un 737 Max de Lion Air au large des côtes indonésiennes, le 29 octobre, un capteur d'angle d'attaque défectueux qui venait d'être installé a envoyé des signaux erronés indiquant que le nez de l'avion était pointé trop haut par rapport à l'air venant en sens inverse. Cela a incité le MCAS (Maneuvering Characteristics Augmentation System) à faire descendre l'avion plus de 20 fois jusqu'à ce que les pilotes perdent le contrôle et qu'il plonge dans la mer de Java, tuant les 189 personnes à bord.

- Le 10 mars, le même système de sécurité sur un 737 Max exploité par Ethiopian Airlines a été activé après qu'un capteur d'angle d'attaque de l'avion soit tombé en panne soudainement au décollage. Après environ six minutes au cours desquelles le MCAS a poussé le nez à plusieurs reprises, l'avion a plongé fortement et s'est écrasé à grande vitesse avec 157 passagers et membres d'équipage.

Mémoire du Xerox Alto

- Système de mémoire du Maxc avec correction d'un bit en erreur et détection de deux bits en erreur. Aucune erreur rapportée dans les fichiers d'erreur.
- Le même système est repris pour le Alto mais avec détection pour 1 bit seulement. Aucune erreur pour 6 mois. Avec le logiciel d'édition plein écran Bravo, de nombreuses erreurs sont apparues subitement.
- Sur le Maxc, une mauvaise configuration faisait que les erreurs de parité n'étaient pas rapportées. De plus, certains patrons de bits causent beaucoup plus d'erreur dans les circuits de mémoire, ce qui est arrivé avec Bravo sur le Alto.
- Sur le Alto 2, avec des circuits de mémoire plus récents, les concepteurs ont remis la correction et la parité. Le système était très fiable mais ils ont découvert éventuellement que 25% de la mémoire n'était pas protégée. Meilleurs circuits? Erreurs non détectées?

Ordinateurs dans la navette spatiale

- Quatre ordinateurs redondants qui exécutent le même logiciel de manière synchrone, avec leurs sorties qui sont synchronisées et comparées 400 fois par seconde. Un cinquième ordinateur de relève qui exécute un autre logiciel programmé de manière indépendante.

- Lors du premier lancement, une erreur a été détectée laissant penser que plusieurs ordinateurs étaient défectueux et le lancement a été retardé.

- C'était un problème de synchronisation.

- Le second logiciel développé de manière indépendante ne protège pas contre les erreurs de spécification.

La fusée Ariane 5

- Le système informatique a été repris tel quel de la fusée Ariane 4. Rien de plus conservateur et sécuritaire!?!
- La valeur d'accélération pour Ariane 4 ne pouvait pas dépasser une certaine valeur maximale.
- Ariane 5 était plus rapide, lorsque la valeur maximale a été dépassée, l'ordinateur a été déclaré fautif et l'ordinateur de relève a été activé.
- L'ordinateur de relève a détecté la même erreur d'accélération trop grande et s'est désactivé.
- Le premier lancement n'a pas bien fonctionné. . .

Commutateurs 4ESS de AT&T, 15 janvier 1990

- Nouvelle version de logiciel installée le décembre précédent sans problème apparent.
- Un problème indépendant en janvier cause une faute qui est détectée sur un commutateur (e.g., A). La procédure corrective est de faire redémarrer le commutateur, ce qui prend 4 à 6 secondes.
- Le système A envoie un message aux autres commutateurs (e.g., B) qu'il n'acceptera plus d'appel car il redémarre.
- Après le redémarrage, l'ancien système envoyait un message pour reprendre les appels, le nouveau recommence directement à initier des appels.
- Le commutateur B, en recevant un appel de A, comprend que A redevient fonctionnel, il réinitialise sa logique interne pour en tenir compte.
- Si B reçoit un autre appel de A pendant qu'il réinitialise sa logique, il devient confus et initie un redémarrage comme A l'avait fait.
- En peu de temps, le réseau de 114 commutateurs n’était plus qu'une cascade sans fin de redémarrages.

Les pannes sont-elles vraiment indépendantes?

- Ordinateurs redondants avec mêmes logiciel/matériel, vulnérabilités, employés, bâtiment. . .
- Les trois capteurs de vitesse du vol Air France 447 de Rio à Paris sont pris dans la glace et donnent une vitesse de 0. L'autopilote de désengage et les pilotes réagissent mal.
- Plusieurs moteurs des avions L-1011 ont fait défaut simultanément. La même équipe avait fait l'entretien de ces moteurs et oublié un joint d'étanchéité torique dans le circuit hydraulique.
- Les 4 moteurs du vol British Airways 9 de Londres à Auckland se sont arrêtés presque simultanément près de Jakarta. La poussière d'un nuage volcanique a causé le problème.