FixMe : détection répartie de défaillances isolées

E. Anceaume[1], E. Le Merrer[2], R. Ludinard[3], B. Sericola[3], G. Straub[2]

[1] *CNRS UMR 6074 IRISA, firstname.name@irisa.fr*
[2] *Technicolor Rennes, firstname.name@technicolor.com*
[3] *INRIA Rennes Bretagne-Atlantique, firstname.name@inria.fr*

Dans cet article, nous nous intéressons au problème de la détection de défaillances dans les réseaux large-échelle. La résolution de ce problème est une des clés de la réduction des coûts de maintenance des opérateurs télécom ou des opérateurs over-the-top. Dans ce contexte, nous présentons une solution distribuée permettant de détecter et distinguer les défaillances isolées des défaillances de groupe en $\mathcal{O}(logN)$ messages, où N est le nombre de noeuds dans le système.

Keywords: Système réparti, réseau logique, détection d'erreur, large échelle

1 Introduction

De nos jours, les équipements connectés sont de plus en plus présents chez l'utilisateur. Ces équipements fournissent des services au travers du réseau. L'opérateur, en déployant ses solutions souhaite être capable de s'assurer du bon fonctionnement de ses applications ou équipements afin d'être capable de réagir rapidement en cas de dysfonctionnement. Cette tâche consiste à surveiller de manière continue le comportement de chacun des équipements. Cependant, la taille de la population à surveiller pose une réelle limite et de ce fait des solutions proactives sont proposées pour pallier ce problème. Ainsi, les opérateurs déploient des centres de support permettant aux utilisateurs détectant un dysfonctionnement d'appeler et d'être pris en charge. Les équipes de support peuvent alors se connecter sur les équipements terminaux et investiguer les causes du dysfonctionnement. Cependant, la notification de dysfonctionnement est laissée à l'appréciation de l'utilisateur. Il arrive donc que le support soit mobilisé alors qu'il n'y a pas de problème réel, mobilisant ainsi inutilement des ressources. Par contre, dans le cas d'une réelle défaillance, la notification n'intervient qu'après que l'utilisateur en ait pris conscience, augmentant ainsi significativement le temps de prise en charge. Il semble donc intéressant de proposer de nouvelles techniques pour la surveillance afin de réduire les délais de prise en charge des notifications et diminuer les détections inutiles réduisant ainsi le coût induit par les tâches de surveillance et de support. Le nombre potentiellement grand d'équipements à surveiller plaide en faveur d'une gestion décentralisée au niveau même des équipements ciblés par la tâche de surveillance. Cependant, l'opérateur ne maîtrise pas nécessairement l'intégralité du réseau d'interconnexion et donc de ce fait ne peut pas collecter d'informations au niveau de celui-ci. Le déport de la tâche de surveillance au niveau des équipements terminaux a déjà été envisagé et des procédures standards [Bro] existent pour la plupart des équipements déployés mais sont désactivées dans la pratique. En effet, les équipements peuvent alors détecter une défaillance alors que celle-ci est en fait située dans le réseau. Dans ce cas, des milliers d'alarmes inutiles sont envoyées à l'opérateur. Les équipements n'étant pas capables de déterminer si l'origine du problème se situe dans l'équipement ou dans le réseau, l'opérateur désactive cette fonctionnalité afin d'éviter l'envoi massif d'alarmes. Il serait donc intéressant d'être capable de distinguer au niveau de l'équipement si la défaillance est isolée ou si elle impacte un grand ensemble de nœuds, on parle alors de défaillance globale. Cette distinction permettrait alors aux seuls nœuds concernés par une défaillance locale d'envoyer une alarme.

Dans cet article, nous proposons une solution visant à faire la distinction entre défaillances isolées et défaillances globales au niveau des équipements terminaux et nous proposons FixMe,

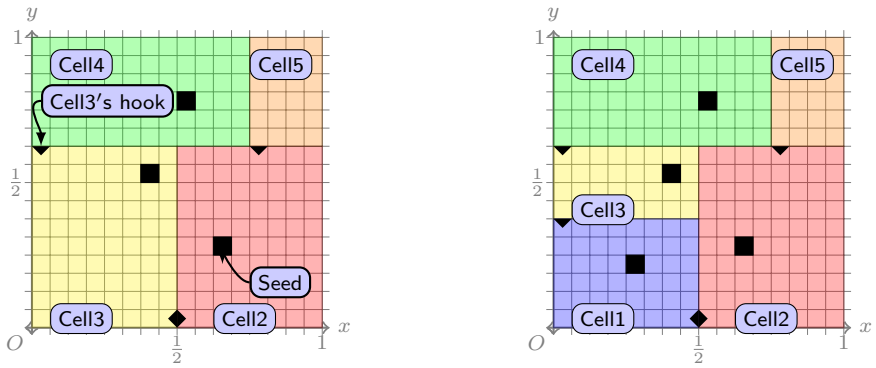

Figure 1: FixMe avant (à gauche) et après (à droite) une opération split

une solution de surveillance répartie auto-organisante, dynamique et passant à l'échelle. Dans la suite de l'article, nous décrivons l'architecture de FixMe ainsi que l'utilisation qui en est faite pour distinguer les défaillances isolées des défaillances globales.

2 Architecture de FixMe

On considère un ensemble de N nœuds connectés au réseau et pouvant communiquer ensemble. Chaque nœud du système possède un identifiant unique obtenu au moyen d'une fonction de hachage et a accès à un ensemble de D services au travers du réseau. Pour chacun de ces services, un nœud p est capable de calculer la valeur à l'instant t de deux fonctions $Q_i(p,t)$ et $A_i(p,t)$. La fonction $Q_i(p,t)$ représente la qualité d'expérience du service i perçue à l'instant t par le nœud p. On peut considérer sans perte de généralité que les valeurs prises par $Q_i(p,t)$ appartiennent à l'intervalle $[0,1]$. On définit alors la position d'un nœud p à l'instant t par $Q(p,t) = (Q_1(p,t), \ldots, Q_D(p,t))$. D'autre part, chaque segment de qualité est séparé en n_i intervalles correspondant à des classes de qualités du service i. La fonction $A_i(p,t)$ est utilisée pour la détection d'anomalies. Celle-ci est alimentée par les valeurs de Q_i des instants précédents. Cette fonction dépend du type de service considéré et peut être une fonction de seuil très simple ou une détection plus évoluée du type Holt-Winters ou Cusum. On considère ici que la sortie de ces fonctions est un booléen permettant de caractériser la variation de qualité comme une anomalie.

De manière similaire à CAN [RFH+01], FixMe organise les nœuds du système dans un espace euclidien torique à D dimensions $[0,1]^D$. Chaque dimension i est découpée en n_i intervalles. Ainsi l'espace initial est partitionné en zones élémentaires appelées *buckets* correspondant au produit cartésien de D intervalles élémentaires. Le partitionnement de l'espace en buckets est statique et est un paramètre du système. FixMe place les nœuds dans cet espace en fonction de la position $Q(p,t) = (Q_1(p,t), \ldots, Q_D(p,t))$ de chacun. Ainsi, quand un nœud entre dans le système, il est inséré dans l'unique bucket correspondant à sa position. Un bucket contenant plus de S_{min} nœuds est appelé *seed*. L'état d'un bucket (seed ou non seed) varie dynamiquement en fonction du nombre de nœuds qu'il contient. FixMe partitionne de manière dynamique l'espace en *cellules* dont le bucket est l'unité élémentaire. Une cellule est un hyper-rectangle de l'espace contenant au plus un seed. Les seeds sont représentés dans la Figure 1 par les carrés noirs et les cellules par les zones de couleurs.

Les cellules sont connectées entre elles par les nœuds présents dans les seeds. Ainsi lorsqu'un nœud perçoit une variation de sa qualité, il effectue une requête de lookup aux nœuds du seed de sa cellule qui l'orientent vers le seed de la cellule la plus proche de sa nouvelle position. En agissant ainsi de proche en proche le nœud arrive alors à atteindre sa nouvelle cellule en $\mathcal{O}(logN)$ messages. Le détail des algorithmes et des complexités se trouve dans [ALS+12].

Lorsqu'un nœud se déplace dans FixMe, il peut s'insérer dans un bucket contenant déjà S_{min} nœuds. Le bucket devient alors un seed et FixMe sépare la cellule en deux nouvelles cellules en fonction des positions des seeds dans la cellule initiale. Chaque seed prévient les cellules voisines

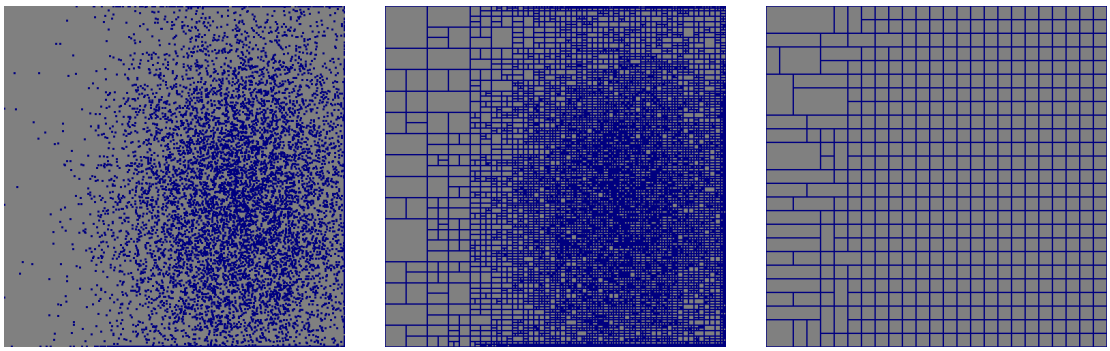

Figure 2: CAN (au centre) et FixMe (à droite) avec une distribution non-uniforme des nœuds(à gauche)

de cette séparation. Le processus de notification des cellules voisines nécessite l'envoi de $\mathcal{O}(D)$ messages. La Figure 1 illustre la séparation de deux cellules : S_{min} nœuds sont présents dans la cellule 3 dans un bucket différent du seed. Lorsqu'un nouveau nœud entre dans ce bucket, celui-ci devient un seed et la cellule 3 se sépare en deux nouvelles cellules 1 et 3 en fonction des positions des seeds.

À l'inverse, lorsqu'un nœud quitte un seed contenant S_{min} nœuds, ce seed redevient un bucket classique et la cellule qui le contient n'a plus de seed. Dans ce cas, les nœuds du bucket déterminent la position du *hook* de la cellule. Le hook de la cellule est le premier bucket, dans l'ordre lexicographique, voisin d'un des coins de la cellule n'appartenant pas à celle-ci. La cellule contenant le hook prend alors en charge la responsabilité de la cellule dépourvue de seed. Les deux cellules peuvent éventuellement être fusionnées en une seule si leur géométrie le permet. Enfin les cellules voisines sont notifiées de la disparition du seed. Cette notification requiert dans le pire des cas l'envoi de $2Dn^{D-1}log(n/2)$ messages avec $n = max_{1\leq i\leq D}n_i$. Les hooks sont représentés dans la Figure 1 par les triangles noirs. On voit par exemple que la cellule 5 est sous la responsabilité du seed de la cellule 2 mais ces cellules ne sont pas géométriquement unifiables. À l'inverse quand la cellule 1 sur la figure de droite disparaît, la cellule 3 prend en charge son espace en unifiant les deux cellules comme on le voit sur la figure de gauche.

Traditionnellement, les systèmes qui utilisent des approches de ce type organisent les nœuds en fonction de leurs identifiants issus d'une fonction de hachage [RFH+01]. Celle-ci ayant la propriété de distribuer uniformément les identifiants sur l'espace d'adressage, le placement qui en résulte est alors également uniforme. Dans le cas présent, nous ne bénéficions pas de cette propriété. En effet, comme les nœuds se placent en fonction de mesures de qualités perçues et que celles-ci n'ont aucune raison d'être uniformément réparties, il se crée potentiellement des endroits du système qui sont fortement peuplés et d'autres dépourvus de nœuds. La Figure 2 illustre ce phénomène : lorsque les nœuds sont placés suivant une répartition non uniforme (à gauche), le comportement de CAN se dégrade (au centre), augmentant ainsi significativement le nombre de zones traversées lors d'un lookup, contrairement à FixMe (à droite) qui se stabilise dès que les cellules ne contiennent plus qu'un bucket, limitant ainsi la longueur des chemins empruntés par un lookup. Afin de conserver les propriétés de passage à l'échelle de FixMe, nous proposons d'organiser les nœuds au sein de chaque seed en utilisant un réseau structuré. De nombreuses solutions de la littérature peuvent être employées, mais il faut cependant garder à l'esprit que des mouvements massifs de nœuds peuvent se produire dans le cas de défaillances réseau. Il est ainsi indispensable de choisir des structures résistantes au churn et qui n'organisent pas les nœuds en fonction de critères géographiques.

3 Défaillances isolées

Nous décrivons maintenant l'algorithme 1 utilisé par FixMe pour déterminer si une défaillance est isolée ou non. Celui-ci est exécuté périodiquement par chaque nœud et repose sur trois tâches

```
   Data :
   T : délai nécessaire pour que chacun des nœuds se soit réplacé dans l'overlay
   τ : seuil pour considérer une défaillance comme isolée
   Output : p est replacé dans l'overlay à l'endroit adéquat et a envoyé une alerte en cas de détection de défaillance
            isolée.
 1 begin
 2     Task 1
 3         r← r+1;
 4         oldposition ← p.bucket ;
 5         newposition ← Q(p,r);
 6         newbucket ← p.lookup(newposition);
 7         if newbucket ≠ p.bucket then
 8             p.leave();
 9             p.join(r,p);
10         end
11     EndTask
12     if ∃i, 1 ≤ i ≤ D, A_i(p, r)=true then
13         Task 2
14             h ← H(oldposition, newposition, r − 1);
15             p.incrementValue(h);
16         EndTask
17         Wait Until T;
18         Task 3
19             n ← p.cell.seed.get(h);
20             if n ≤ τ then
21                 send FixMe msg to Management Operator;
22             end
23         EndTask
24     end
25 end
```

Algorithme 1 : p.updatePosition(r :round)

élémentaires : déplacement, diagnostique et alerte.

Soit le nœud p exécutant l'algorithme pendant la ronde r ; ce nœud calcule sa nouvelle position $Q(p, r)$. Soient b_r, c_r, s_r représentant respectivement le bucket auquel appartient p à l'instant r, la cellule en charge de b_r et le seed en charge de c_r. Si b_r est différent de b_{r-1} alors p quitte b_{r-1} et rejoint b_r. Si la variation de qualité est suffisamment importante pour être considérée comme une anomalie, alors il existe un service i tel que $A_i(p, r) = \texttt{True}$. Dans ce cas, p cherche à déterminer si cette anomalie est isolée ou non en exécutant la tâche 2. Les nœuds de s_r sont organisés en DHT suivant leurs identifiants. Le nœud p calcule une clé aléatoire $\mathcal{H}(b_{r-1}, b_r, r-1)$ où $\mathcal{H}$ est une fonction de hachage. Ensuite p contacte le nœud n de s_r en charge de cette clé pour incrémenter la valeur d'un compteur initialisé à zéro au début de la ronde (ligne 15 de l'algorithme). Après un délai T, nécessaire au replacement de chaque nœud du système, p contacte de nouveau le nœud n pour obtenir la valeur du compteur (ligne 19). Si cette valeur est inférieure à τ, paramètre du système, alors p considère cette défaillance comme isolée et envoie une alerte à l'opérateur.

Nous montrons dans [ALS+12] que l'algorithme 1 permet de détecter les défaillances isolées de manière répartie avec un coût de $\mathcal{O}(logN)$ messages où N est le nombre de nœuds dans le système. Nous envisageons à présent d'évaluer la performance et la précision des algorithmes proposés au moyen d'une étude probabiliste.

Références

[ALS+12] E. Anceaume, R. Ludinard, B. Sericola, E. Le Merrer, and G. Straub. FixMe : A Self-organizing Isolated Anomaly Detection Architecture for Large Scale Distributed Systems. In *Proceedings of the 16th International Conference On Principles Of Distributed Systems (OPODIS)*, page 12, Rome, Italie, December 2012.

[Bro] Broadband Forum. TR-069 CPE WAN Management Protocol Issue 1, Amend.4, 2011.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.