Routage distribué de flux RSS

Ngom Bassirou* — Sarr Idrissa** — Ndiaye Samba** — Gueye Modou***

* Département Mathématiques Informatique
Université Cheikh Anta Diop
Dakar
Senegal
bassirou83.ngom@ucad.edu.sn

** Département Mathématiques Informatique
Université Cheikh Anta Diop
Dakar
Senegal
prenom.nom@ucad.edu.sn

*** Département Mathématiques Informatique
Université Cheikh Anta Diop
Dakar
Senegal
gmodou@gmail.com

RÉSUMÉ. Actuellement le nombre de sources RSS est en très forte croissance. Les besoins des utilisateurs évoluent en conséquence. Ainsi, la charge des demandes d'actualisation de flux générée par les applications devient considérable et pas toujours nécessaire. Cela peut entraîner le ralentissement des systèmes ou induire des surcharges de communication. Les solutions existantes pour gérer les demandes d'actualisation de ces flux s'appuient sur des architectures centralisées ce qui ne favorise pas la scalabilité et la disponibilité des flux. Nous proposons une nouvelle solution pour réduire la charge soumise aux sources RSS. Le protocole s'appuie sur une diffusion ciblée des items à transmettre aux abonnés à travers un réseau structuré de pairs. Il permet aussi aux abonnés de faire des recherches sélectives basées sur des mots clés. Nous avons implémenté notre proposition avec le simulateur Peersim pour étudier sa faisabilité.

ABSTRACT. Currently the number of RSS feeds is very strong growth and user needs evolve accordingly. Thus, the load of requests to update RSS feeds generated by applications is enormous and not always necessary. This can result to breakdown of the systems or leads to communication overload. Existing solutions to manage requests to RSS feed updates are based on centralized architectures, thus, do not support the scalability and RSS feed availability. We propose a new solution to reduce the load submitted to the RSS feed providers. The protocol is based on a targeted distribution of items to be transmitted to subscribers through a structured network of peers. It also allows subscribers to do selective keyword-based requests. Moreover, we implemented our proposal with Peersim simulator to study its feasibility.

MOTS-CLÉS : RSS, Routage, syndication de contenu, P2P, polling

KEYWORDS : RSS, Routing, Content syndication, P2P, polling

1. Introduction

Les utilisations importantes et diversifiées du web (*i.e. e*-commerce, réseaux sociaux, ...) requièrent des mises à jour fréquentes de contenu. L'importance de la quantité des informations et leur caractère dynamique rendent impraticable toute forme de mises à jour manuelle faites par un administrateur de site web. C'est ainsi, que de nouvelles technologies ont vu le jour pour faciliter la collecte et la mise en ligne d'informations provenant de diverses sources. Ces technologies offrent la possibilité d'obtenir du contenu par aggrégation de différentes sources d'informations mais aussi de donner l'autorisation à d'autres sites web d'user des mêmes informations, on parle de syndication de contenu web. L'échange entre fournisseurs (sites d'actualités, blogs, ...) et clients peut se faire via des protocoles standards comme RSS (*Really Simple Syndication*). RSS désigne une famille de formats XML utilisée pour la syndication de contenu web. Le mode de fonctionnement actuel des flux RSS repose sur la scrutation périodique des sources par les applications clientes. Les sites web qui proposent des flux doivent donc répondre à plusieurs requêtes d'utilisateurs durant la journée et ceci même s'ils ne disposent pas de nouveau contenu ou de contenu récent. Deux problèmes peuvent être soulignés dans le fonctionnement de RSS :

– Le format des messages est statique. Lorsqu'un client scrute un flux RSS, toutes les entrées du flux lui sont renvoyées même si certaines ne sont pas nouvelles pour lui. D'où il y a une redondance d'informations et une surcharge du réseau.

– Le polling des clients est permanent et désorganisé. De ce fait, les demandes d'actualisation de flux faites par les applications clientes génèrent une charge considérable et pas toujours nécessaire. Ceci entraîne le ralentissement des systèmes. De plus, les solutions existantes pour gérer les demandes d'actualisation de flux s'appuient sur des architectures centralisées qui ne favorisent pas le passage à l'échelle et la disponibilité des flux.

Dans ce document, nous proposons une nouvelle approche de syndication web basée sur une architecture distribuée. Notre solution réduit la charge au niveau des fournisseurs de flux. Il permet aussi de répondre efficacement aux besoins des utilisateurs grâce à un système de recherche de flux basée sur des thèmes.

Nous avons mis en place un environnement pair-à-pair pour distribuer la charge au niveau des fournisseurs de flux. Notre solution utilise un algorithme que nous avons appelé "polling between server" : le polling est contrôlé et reste limité au niveau des fournisseurs. A cela s'ajoute une réplication des flux sur plusieurs fournisseurs pour assurer la disponibilité des flux. Pour valider notre approche, nous avons implémenté notre proposition avec le simulateur Peersim.

La suite de ce document est structuré comme suit. La Section 1.1 présente l'architecture et les composants de notre solution. La Section 2 détaille notre protocole de recherche et/ou de routage des flux alors que la Section 3 expose la scrutation entre serveurs. La Section 4 décrit le gestion de la disponibilité du système. La Section 5 présente les résultats obtenus avec notre simulation alors que la Section 6 relate les travaux connexes et la Section 7 conclut.

1.1. Architecture

Nous supposons que les flux sont valides selon la norme RSS ou Atom et que les types de flux sont limités et prédéfinis (ne dépassent pas vingt). Nous ne nous intéressons pas à la manière dont les flux sont créés. L'architecture (*cf.* Figure 1) est composée d'un

ensemble de noeuds regroupés autour d'un anneau logique et utilise des notions de groupe logique comme dans JuxMem [1]. Il comporte trois types d'infrastructures : nous avons d'un coté les fournisseurs de flux que nous allons appeler par la suite RSS Peer (RP) et de l'autre côté les utilisateurs qui seront dénommés ici par RSS Client (RC). Entre les deux, nous avons les gestionnaires qui sont des RP particuliers dont le rôle est de coordonner l'offre de flux par les RP et leur demande par les RC. Nous avons classé les flux selon leur type et identifié chacun par l'url à partir duquel il est accessible. Chaque flux est représenté alors par une entrée sous forme de couple (Title, Link). Nous allons repartir les RP dans des groupes RSS "RSS Group" qui sont des structures logiques regroupant les RP qui publient des flux d'un type donné et les clients intéressés par ce type de flux. Ces groupes sont managés par des GRP.

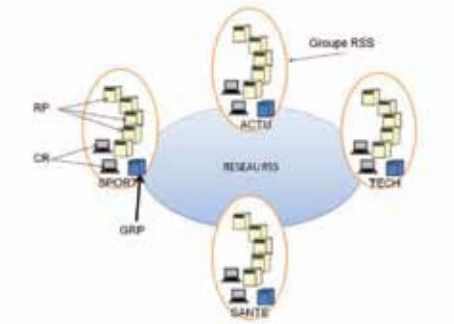

Figure 1. *Architecture*

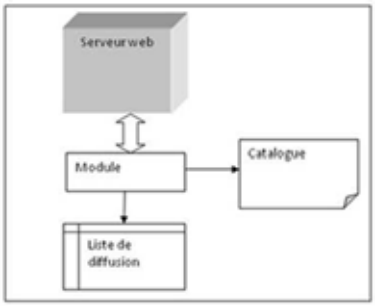

Figure 2. *Structure d'un RSSPeer (RP)*

1.2. Rôle des composants de l'architecture

Nous allons à présent définir le role des différents composants de notre système.

– **Les RSS Peers (RP) :** Le RP permet de connaître les autres RP du groupe en maintenant un catalogue contenant les adresses des RP du groupe et leur flux . Chaque RP scrute les autres sources du groupe et cachent les dernières entrées diffusées par un serveur mais garde une historique complète de ses propres flux.Il maintient une table de diffusion pour chaque flux. Cette table contient les adresses des clients qui sont connectés sur lui.

– **Les gestionnaires des RSS Peers (GRP) :** Les GRP sont des RP particuliers qui gèrent le groupe en permettant l'insertion, l'authentification des RC, la localisation des fournisseurs de flux et la mise à jour des catalogues. A la différence des catalogues des autres RP celui du GRP contient en plus le nombre de clients connectés sur chaque RP.

– **Les Clients RSS (RC) :** Les RC appartiennent à un groupe RSS contenant le type de flux désiré et reçoivent régulièrement les mises à jour de ce dernier. Ils possèdent une liste de mots clés caractérisant les flux qui les intéressent et les GrId des groupes dans lesquels ils étaient connectés la dernière fois. Cette liste peut se présenter sous la forme suivante *(thème, fournisseur, item, date de publication)*, où *thème* désigne les types de flux voulu, *fournisseur* les fournisseurs de ces flux, *item* un ou plusieurs mots donnant des renseignements sur l'entrée voulue, *date de publication* permet de dire que le client est intéressé par les entrées publiées à partir de la date indiquée. Par exemple un client peut avoir la liste suivante : (news, lemonde, " Europe", " 05/07/2010 8h ").

2. Algorithme de gestion et de diffusin des flux

Les spécifications sur les quelles s'appuie notre protocole sont :

– Un Client RSS ne souhaite pas recevoir les actualisations en continu mais seulement lorsqu'il est disponible pour les consulter, c'est-à-dire s'il est connecté au réseau RSS ;

– Un Client RSS souhaite recevoir seulement les actualisations les plus récentes (ex. celles produites il y a moins d'un jour) et correspondant à ses thèmes (liste de mots-clés) ;

– La demande provenant d'un Client RSS peut être traitée sans interroger la source.

Ci-dessous nous détaillerons l'algorithme qui permet de traiter cette demande de fournisseur.

2.1. Recherche de fournisseur

2.1.1. Recherche locale

Notre objectif est de trouver le fournisseur le moins chargé et disposant des entrées recherchées. Le principe est très simple et se déroule comme suit :

• (a) le GRP qui reçoit la demande et qui constate que la recherche porte sur un flux ayant le type du groupe vérifie la date indiquée par le client ;

• (b) si la date est supérieure à la date de fraicheur, la requête est transmise au RP fournisseur du flux. Si ce dernier n'est pas disponible le client est avisé de l'indisponibilité du flux ;

• (c) sinon la requête est envoyée au RP ayant le moins de clients ;

• (d) le RP accuse réception du message ;

• (e) si l'accusé est reçu, le GRP incrémente le compteur du RP choisi et l'algorithme se termine ;

• (f) sinon on reprend à partir de (c) ;

• (g) le RP choisit ajoute le client dans sa liste de diffusion.

Nous remarquons que si la demande contient des termes trop contraignants (exemple : date très ancienne) il y a de faible chance d'avoir une réponse car un seul fournisseur peut répondre. Mais si l'entrée est récente, elle est repliquée dans les autres RP et la requête sera traitée par le serveur le moins chargé.

2.1.2. Recherche globale

Pour faire la recherche au niveau gloabl nous avons l'algorithme suivant :

• (a) le gestionnaire qui reçoit une requête de la part d'un client et qui s'aperçoit qu'il ne gère pas le type demandé le transmet à son successeur et son prédécesseur en y joignant un compteur C = (N/2) - 1, N est le nombre de groupes présents dans le réseau, ceci permet de diviser le réseau en deux parties et d'exécuter des recherches en parallèles ;

• (b) chaque GRP qui reçoit une demande transmise par un GRP vérifie s'il peut la traiter ;

• (c) si oui, il initie une recherche locale et répond au client ;

• (d) sinon, il transmet la requête au gestionnaire du groupe qui le succède dans le sens de propagation de la requête en décrémentant C ;

• (e) la recherche se termine C = 0. Le noeud ayant le compteur nul avise le client de l'insatisfaction de sa demande.

Cet algorithme offre une bonne complexité en termes de messages car nous avons au pire des cas une réponse avec ((*N*/2) +1) messages en plus de la recherche locale.

3. Scrutation entre serveurs

Notre approche se base sur la scrutation entre les serveurs fournisseurs de flux. Ceci dans le but de réduire la charge de ces derniers et de pouvoir contrôler le polling, véritable talon d'achille des flux RSS. Pour assurer le fonctionnement de la scrutation entre serveurs, nous utilisons une procédure classique connue au niveau des systèmes publication-abonnement qui est la notification. A chaque apparition d'une nouvelle entrée le RP envoie un message *notify* aux autres RP du groupe et ces derniers scrutent le flux. Avant de stocker les entées, le flux récupéré est filtré, seules les entrées apparues après la date de fraicheur sont retenues. Ainsi nous voyons que la fréquence de polling dépend de la fréquence des mises à jour. Un serveur ayant un fort taux de mise à jour se voit scruté plus fréquemment qu'un autre qui en a moins et par conséquent verra ses entrées plus disponibles dans le réseau. Ceci constitue un point fort pour la syndication de contenu car au lieu de surcharger les fournisseurs et de les amener à appliquer des politiques pour limiter la diffusion, et de nuire les principes du RSS, nous les poussons à fournir plus pour mieux bénéficier du système.

La synchronisation entre les serveurs peut évidemment induire une surcharge. En effet au pire des cas nous aurons une surcharge de *n(n-1)* requêtes poll sur chaque RP ; n étant le nombre de RP dans le groupe. Cette situation a lieu uniquement si tous les *n* RP publient en même temps une mise à jour et que les *n-1* autres RP viennent le scruter. Comparé au poll actuel, les n(n-1) requêtes poll peuvent être négligées vis-à-vis des millions de requêtes poll que ce même RP est obligé de subir à des fréquences rapprochées (même s'il n'y avait de mise à jour).

4. Maintenance des gestionnaires

Pour prendre en compte la volatitilité des gestionnaires, nous avons répliqué le rôle du gestionnaire sur un autre RP qui devient GRP secondaire (GRPS) en cas d'indisponibilité du GRP. Chaque GRP choisit parmi les RP disponibles, celui le moins chargé qu'il nomme GRPS. Il envoie régulièrement l'état du réseau à ce dernier qui répond par un acquittement. Ainsi, si le GRP tombe en panne le GRPS peut prendre le relais sans perdre trop d'informations. Pour assurer qu'il y ait toujours un GRP disponible, la démarche suivante est adoptée :

• après une période ***r***, si le GRPS ne reçoit pas de rafraîchissement de la part du GRP, il peut se déclarer GRP en informant aux autres noeuds du groupe de son nouveau rôle ("INTENTION") et la date du dernier rafraîchissement qu'il a reçu du GRP ;

• si un noeud reçoit cette "INTENTION", il compare la date envoyée par le prétentieux nouveau GRP (jusqu'ici GRPS) et celle de sa dernière interaction avec le GRP suspecté en panne ;

• si la date de cette dernière interaction est supérieure à celle envoyée par le prétentieux nouveau GRP (ancien GRPS), le noeud renvoie un message au prétentieux nouveau GRP en lui donnant la date de sa dernière interaction avec l'ancien GRP ;

• si aucun message n'est envoyé au prétentieux nouveau GRP, cela veut dire qu'aucun noeud du groupe n'a pu contacter l'ancien GRP après la date donnée par le prétentieux nouveau GRP, et donc on peut supposer que l'ancien GRP est en panne. Par conséquent le prétentieux nouveau GRP envoie un message de confirmation ("CONFIRMATION") aux autres noeuds du groupe qui, désormais, le considèrent comme GRP ;

• si toutefois, un message a été envoyé au prétentieux nouveau GRP, ce dernier diffère d'une période ***r***, sa tentative de devenir GRP. Si à la fin de cette période ***r***, il ne reçoit toujours pas de rafraîchissement, il réitère le processus décrit précédemment ;

• après ***k*** tentatives soldées par des échecs, nous concluons qu'il y a une panne de communication entre le GRPS et le GRP et donc le GRPS arrête d'émettre ses intentions et se considère comme un RP simple.
Si le GRP ne reçoit plus d'acquittement venant du GRPS pendant une période ***r***, deux cas sont possibles : soit il y a une panne de communication entre les deux entités, soit le GRPS est en panne. On procède de la même manière pour détecter la disparition du GRPS. Si le GRP reçoit une réponse lui notifiant que l'ancien GRPS est joignable, il diffère son intention de ***r***. S'il ne reçoit pas de message ou après k tentatives il élit un nouveau GRPS.

5. Expérimentation et validation

L'objet de cette section est d'évaluer l'approche que nous avons présentée dans les sections précédentes. Pour évaluer les performances de notre sytèeme, nous allons :

• montrer la rapidité de notre protocole comparé à la solution naïve où chaque fournisseur dispose ses propres flux qui sont scrutés régulièrement par les clients ;

• montrer le comportement de la charge du système dans le temps et de vérifier s'il y a bien un équilibrage de charge entre les serveurs ;

• prouver le passage à l'échelle de notre système.

5.1. Apport de notre solution

Avec Peersim [2], nous essayons d'abord de comparer notre protocole avec le fonctionnement actuel des flux RSS c'est-à-dire le fonctionnement centralisé. Pour ce faire nous exposons les deux systèmes dans les mêmes conditions : nous avons fixé le nombre de serveur à 80 et le nombre de client à 100, chacun émettant 4 requêtes par cycle. Nous observons le système pendant un certain nombre de cycles (30 cycles). Nous mesurons la variation de la charge des deux systèmes qui correspond au nombre total de requêtes en cours. Nous représentons les résultats obtenus dans la Figure 3. Nous remarquons que la charge des serveurs est moins importante avec notre protocole qu'avec le fonctionnement centralisé des flux RSS. Ceci s'explique par le fait que si un serveur ne fournit pas de mises à jour récentes, il participe à la diffusion des derniers items des autres fournisseurs, ce qui réduit la charge moyenne des fournisseurs. Ce faisant, nous pouvons dire que la réduction de la charge au niveau des serveurs est assurée dans notre protocole. Par ailleurs on voit sur la Figure 3, que près de 200 requêtes restes dans le système sur un total de 12000 requêtes (400 requêtes par cycle au bout de 30 cycles) avec notre protocole. Nous constatons qu'en 30 cycles et dans les mêmes conditions expérimentales, notre proposition a répondu à prés de 98% des demandes contre 90% pour le régime actuel ; ainsi notre protocole offre un temps de réponse moyen très favorable.

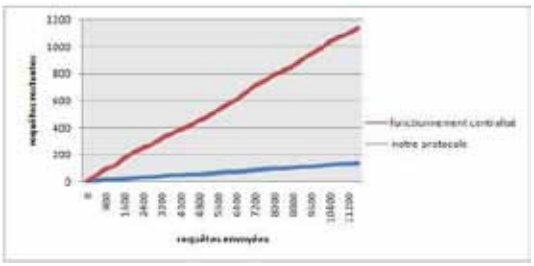

Figure 3. *Evaluation de la charge des serveurs RSS : nombre total de requêtes en cours au niveau des serveurs.*

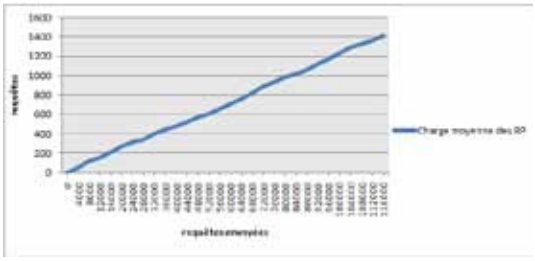

Figure 4. *Variation du nombre de requêtes en attente dans les serveurs en fonction du nombre de requêtes.*

5.2. Passage à l'échelle

Dans un second temps, nous essayons de montrer que notre approche favorise le passage à l'échelle. Ainsi nous soumettons une charge de 4000 requêtes par cycle (chacun des 100 clients émet 40 requêtes) et nous observons la variation de la charge au niveau des serveurs. Nous observons dans la Figure 4, que l'augmentationt progressive du nombre de requêtes dans les serveurs, ne dégrade pas les performances de notre protocole. En effet nous constatons qu'avec 16000 émises, le nombre de requêtes en attentes est de 210 ; ce qui correspond à un pourcentage de 1,3%. Le même résultat est obtenu avec 116000 requêtes émises. Ceci confirme la tendance de la Figure 3 où on avait 100 requêtes par cycles. Ce résultat prouve que le système parvient à fonctionner avec un nombre important de noeud. Ainsi nous pouvons conclure qu'avec notre protocole l'augmentation du nombre de clients n'induit pas de surcharge , donc favorise le passage à l'échelle.

6. Travaux connexes

Les solutions existantes les plus remarquables tentent de décentraliser la gestion des flux et de diminuer la surcharge au niveau des fournisseurs et du réseau. Pour ce faire, certains ont opté pour l'amélioration du polling en compressant les données à transmettre ou en utilisant d'autres moyens tels que les TTL ou les éléments <skipdate> et <skiphour>[3, 4]. Malheureusement le caractère imprévisible de la production des flux RSS fait que cette approche n'a pas eu beaucoup de succès. D'autres ont tenté d'alléger la charge des serveurs en mettant les flux sur un serveur central dédié connu sous le nom "outsourced aggregation". Les solutions d'amélioration apportées sur le fonctionnement centralisé de RSS sont toutes confrontées d'une manière ou d'une autre à des problèmes de passage à l'échelle. C'est ce qui a motivé les chercheurs à trouver d'autres moyens de communication pour diffuser les flux. Par exemple, FeedTree [5, 6] a été proposé dans l'optique de décentraliser la gestion des flux. FeedTree est conçu en utilisant SCRIBE [7] qui est un système de communication par groupe basé sur PASTRY [8]. En dehors des problèmes de bootstrap et ceux des arbres de diffusion (fraîcheur des mises à jour), FEEDTREE reste toujours confronté à certaines difficultés inhérentes au bon fonctionnement des flux RSS tels la superfluidité des messages et le manque de contrôle au niveau des entrées "items " voulues.

FeedEx [9] propose une solution qui se base sur le protocole XML-RPC [10]. FeedEx a pallié certaines déficiences rencontrées dans les processus de distribution des flux RSS.

Mais nous constatons que la recherche d'un flux peut prendre plus de temps que nécessaire : car la demande est itérative (peut atteindre une complexité de O(n)) et n'assure pas toujours de réponse.

7. Conclusion

Dans ce document, nous avons apporté un nouvelle architecture pour le routage des flux RSS partant du fait que les solutions existantes ou proposées ne répondaient pas efficacement à certains besoins. Ce qui avait conduit certains fournisseurs à limiter leur flux ou à appliquer des politiques de défense contre le polling "agressif" des utilisateurs. Nous avons proposé une solution à ces problèmes en éliminant le polling des utilisateurs et en permettant à ces derniers de faire des recherches à base de mot-clès. Nos expérimentations montrent que, comparée à la solution centralisée, notre approche est plus performante en termes de nombre de messages envoyés. Pour satisfaire une recherche, nous avons un nombre de message qui tourne autour de N+n+4 (N étant le nombre de groupes et n le nombre de fournisseurs dans le groupe choisi). Notre protocole permet la réduction de la charge des serveurs due au polling. Il permet aussi la récupération d'un sous ensemble d'items d'un flux donné ce qui résoud le problème de la superfluidité. Dans nos futurs travaux, nous comptons déployer un prototype expérimental dans PlanetLab et comparer notre solution avec celles de Feedtree et FeedEx.

8. Bibliographie

[1] http ://gforge.inria.fr/projects/juxmem/

[2] http ://sourceforge.net/projects/peersim/

[3] R. Scoble. A theory on why RSS traffic is growing out of control.http ://radio.weblogs.com/0001011/2004/09/08.html#a8200, Sept. 2004.

[4] R.C. Morin. HowTo RSS Feed State. http ://www.kbcafe.com/rss/rssfeedstate.html, Sept.2004.

[5] http ://www.feedtree.net/

[6] D.Sandler, A.Mislove, A. Post, P.Druschel. FeedTree : Sharing Web micronews with peer-to-peer event notification, In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*. Ithaca, NY, February 2005.

[7] M. CASTRO, P. DRUSCHEL, A.-M. KERMARREC, A. ROWSTRON, « SCRIBE : A large-scale and decentralized application-level multicast infrastructure », *IEEE JSAC*, 20(8), October 2002.

[8] A. Rowstron and P. Druschel, Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems,*In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*. Heidelberg, Germany, Nov 2001.

[9] Jun, S. and Ahamad, M. 2006. FeedEx : collaborative exchange of news feeds. In *Proceedings of the 15th International Conference on World Wide Web (Edinburgh, Scotland, May 23 - 26, 2006). WWW '06. ACM Press*, New York, NY, 113-122.

[10] D. Winer. XML-RPC specification. http ://xmlrpc.com/spec

[11] G. MÜHL, « Large-Scale Content-Based Publish/Subscribe Systems », *PhD thesis*, Darmstadt University of Technology, 2002.