Etude Nosql

Restitution tests Riak

Auteur : Rosa Mopolo

Sommaire

- Principales caractéristiques de Riak
- Tests réalisés
- Environnement de test
- Caractéristiques de Riak
- Installation de Riak
- Chargement des données
- Test de charge
- Ajout d'un nouveau node sur le cluster
- Augmentation du facteur de réplication
- Simulation panne réseau
- Coupure de site
- Backup/restore
- Crash de node
- Conclusion
- Riak dans le futur

Principales caractéristiques de Riak (1/3)

- **Envoi de requêtes avec Riak**
  - Les requêtes sur Riak se font sous forme de requêtes HTTP :
    - Insertion/modification :
      - curl -v -X PUT -d 'this is a test' -H "Content-type: text/plain" http://*adresse_host*:*port_host*/riak/*lebucket*/*lekey*
      - curl -v -X PUT -d '{"bar":"bar"}' -H "Content-type: application/json" http://*adresse_host*:*port_host*/buckets/*lebucket*/keys/*lakey*
    - Lecture :
      - curl -v http://*adresse_host*:*port_host*/riak/*lebucket*/*lakey*
      - curl -v http://*adresse_host*:*port_host*/buckets/*lebucket*/keys/*lakey*
    - Suppression :
      - curl -v -X DELETE http://*adresse_host*:*port_host*/riak/*lebucket*/*lakey*
      - curl -v -X DELETE http://*adresse_host*:*port_host*/buckets/*lebucket*/keys/*lakey*

Principales caractéristiques de Riak (2/3)

- **Moteurs de stockage**
  - Riak propose 4 moteurs de stockage au choix :

    Bitcask : stocke les données (clé/valeur) dans des hash tables structurées comme des logs

    ➔ Accès très rapide aux données

    ➔ Stockage par défaut, le meilleur lorsque l'on veut stocker des données simples en ayant l'accès le plus rapide possible aux données

    LevelDB : utiliser pour stocker des données ayant des liens les une entre les autres

    ➔ Architecture de stockage proche de celle de BigTable avec le couple memtable/sstable

    ➔ Ne possède pas les limitations en RAM de Bitcask

    ➔ L'accès au lecture peut ralentir lorsqu'il y a trop de niveaux à chercher

Principales caractéristiques de Riak (3/3)

Memory : pour pouvoir stocker toutes les données uniquement dans des tables en mémoire

➔ Utile pour tester les clusters Riak

➔ Les données ne sont jamais stockées sur le disque ou un autre type de stockage

Multi : pour utiliser plusieurs façons de stocker les données en faisant des combinaisons entre les 3 modes précédemment cités

➔ Utile si on veut utiliser différents moteurs de stockage pour différents buckets

➔ Utile si on veut utiliser de façons différentes un moteur de stockage pour différents buckets

- **Réplication**
  - Riak dispose d'un cluster où chaque node possède le même rôle
  - Chaque nœud peut rejoindre ou quitter le cluster en une commande

Quelques fonctionnalités intéressantes de Riak

- **Possibilité de lier des données entre elles avec des liens simple ou des liens « marchant »**
- **Possibilité d'effectuer du MapReduce afin d'exécuter des requêtes complexes**
- **Possibilité d'effectuer des recherches avec la commande *search-cmd***
- **Possibilité de poser des index secondaires**
- **Utilisation d'une horloge vectorielle**

Tests réalisés

- **Les tests réalisés correspondent à des charges se rapprochant de celles de PnS. Pour des raisons évidentes (confidentialité), les jeux d'essai utilisés ne sont pas les données réelles de PnS.**

- **Toutefois, la volumétrie de données de tests (nombre de tables, nombre de champs par table, longueur des champs, nombre de champs Null/not Null) est représentative de la volumétrie de PnS**

- **Les données constituant les jeux d'essai (chaines de caractères, nombres, dates, images) ont été générées de manière aléatoire (seule contrainte : respecter la volumétrie de PnS)**

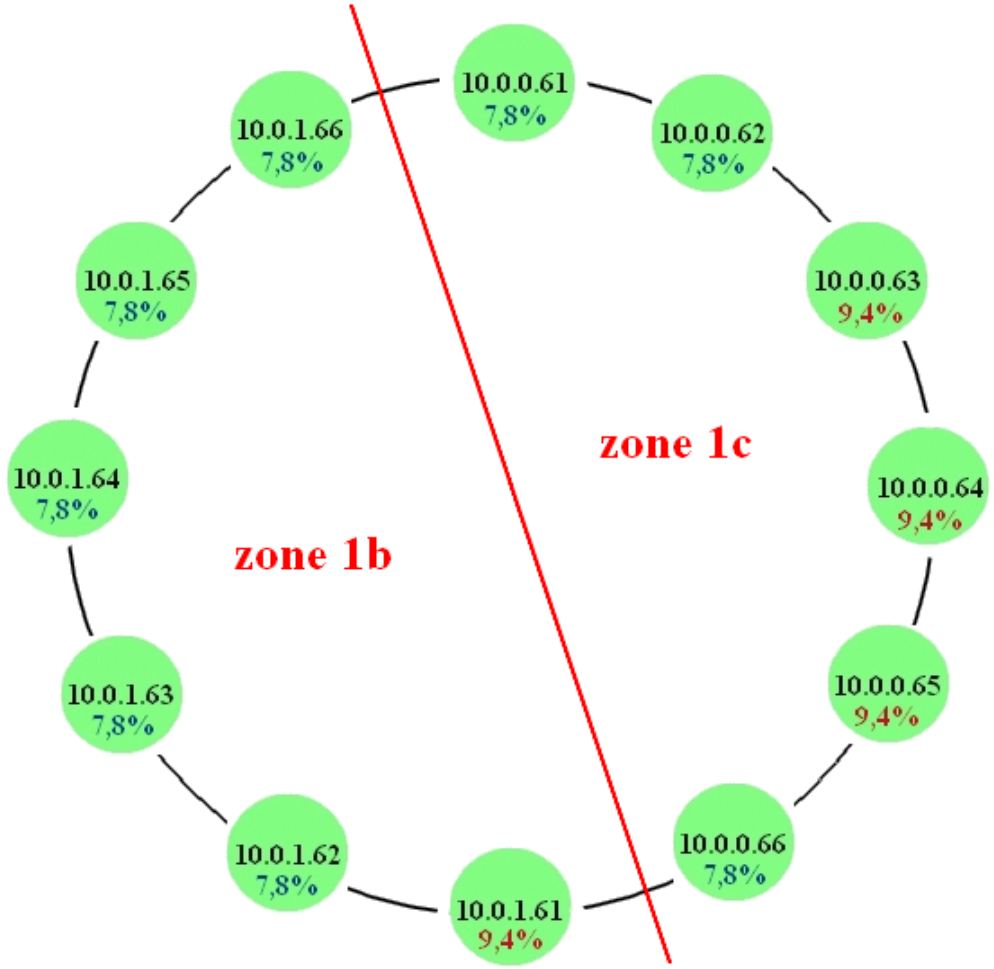
- **Riak a été testé sur la partie « lot MMC ».**

Environnement de test (1/2)

- **La plateforme utilisée chez AWS est constituée de plusieurs serveurs Linux :**
  - 12 *instances larges* réparties sur 2 zones de disponibilité différentes, utilisées comme serveur Riak. Chaque instance dispose de 7,5 Go de RAM, 4UC (4 cores avec 2 UC/core), 850 Go de stockage disque, plateforme 64 bits, Ubuntu 11.10
  - 1 *instance extra_large* (15 Go de mémoire, 8 UC (4 cores avec 2 UC/core), 1690 Go de stockage, plateforme 64 bits, Ubuntu 11.10, utilisée comme injecteur de donnée

Environnement de test (2/2)

- **Les données sont réparties sur les 12 nodes Riak selon le schéma ci-dessous.**
- **Jmeter est utilisé pour envoyer les requêtes vers Riak**



Installation de Riak

- **Installation simple :**
  - Télécharger Riak (la version utilisée lors de ces tests est la 1.1.2)
  - Installer le package
    - Si on utilise Ubuntu 11.10, il faut également installer le package libss10.9.8
  - Et ça marche ☺
  - Riak peut fonctionner sur Linux, BSD, Mac OS X et Solaris.
- **Configuration :**
  - Modifier les fichiers de configurations
    - app.config : modifier les chemins, les hosts, les ports et mettre le nombre de réplication par défaut pour les buckets
    - vm.args : spécifier nom de la machine Riak et son adresse
    - /usr/sbin/riak : seulement si on veut modifier le chemin où toutes les data seront stockées
  - Pour faire entrer un nœud dans le cluster
    - riak-admin join new_name@new_host

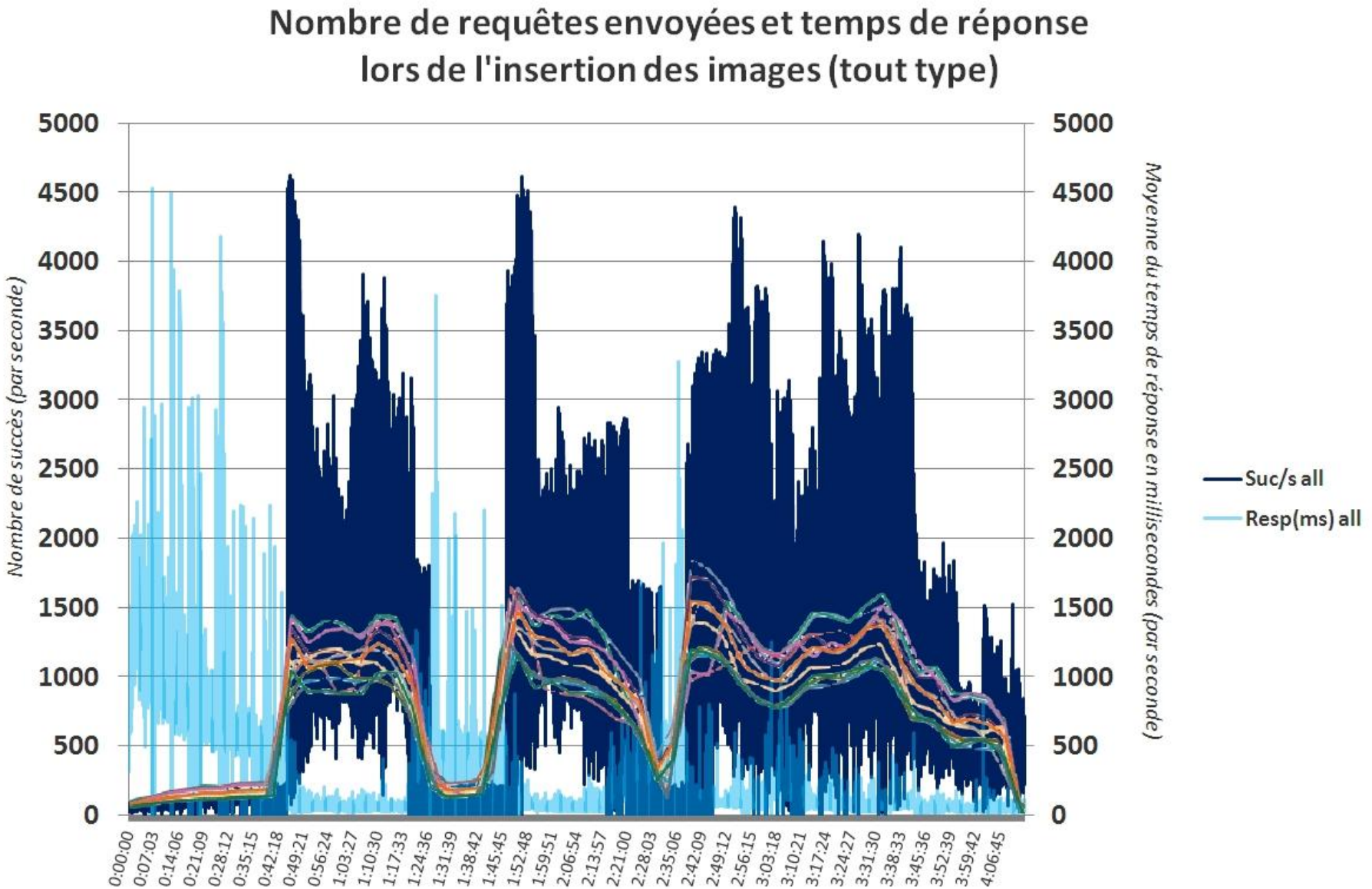
Simple à configurer.

Chargement des données (1/3)

- **4 millions d'ISE (soit 16 millions d'images) ont été insérés en consistance 3 avec le requête suivante :**
  - curl -X PUT http://${HOST}:${PORT}/buckets/${ise}/keys/${imgmax}?w=3 -H “Content-type: image/jpeg” --data-binary@*mon_image.jpg*
- **Le tableau ci-dessous indique les performances obtenues :**

	Durée totale de l’insertion	Moyenne requêtes/seconde	Moyenne temps de réponse (ms)	Temps de réponse maximum (ms) pour une disponibilité de 99,99%
Maxi	**4h 13m 36s**	**262**	**73**	**3 240**
Medium	**3h 54m 25s**	**284**	**67**	**3 070**
Mini	**3h 41m 50s**	**300**	**63**	**3 030**
Micro	**3h 38m 08s**	**305**	**62**	**3 080**

Chargement des données (2/3)



Chargement des données (3/3)

- **4 millions d'ISE (soit 16 millions d'images) ont été insérés sous la forme suivante:**
  - curl -X PUT http://${HOST}:${PORT}/buckets/${ise}/keys/${imgmax}?w=3 -H “Content-type: image/jpeg” --data-binary@*mon_image.jpg*
- **Le tableau ci-dessous indique les performances obtenues :**

Adresse des instances	Tailles des données en Go
10.0.0.61, 10.0.0.62, 10.0.1.61, 10.0.1.62	**74**
10.0.0.63, 10.0.0.64, 10.0.0.65, 10.0.0.66 10.0.1.63, 10.0.1.64, 10.0.1.65, 10.0.1.66	**62**

Conclusions du test de chargement des données

- **Plus l'image l'on insère est petite, plus les temps de réponses vont être faibles.**
- **Plus l'image l'on insère est petite, plus le nombre d'images insérées par seconde augmente**
- **Plus le temps de réponse est élevé, plus le nombre de requêtes envoyées est faible**

Test de Charge (1/4)

- **Nous avons fait un test de charges sur les read et les update où les requêtes seront envoyées en parallèle pour les 4 tailles d'images différentes avec :**
  - Une consistance = 2 sur les read
  - Une consistance = 3 sur les write

- **On démarre les tests avec une charge faible**

Test de Charge – organisation du test (2/4)

- **Les paliers prévus :**

Palier	Nb req/sec en read	Nb req/sec en update
1h	10	1
2h	60	51
3h	110	101
4h	160	151
5h	210	201
6h	260	251
7h	310	301
8h	360	351
9h	410	401
10h	460	451

Test de Charge – Résultats (3/4)

- En read

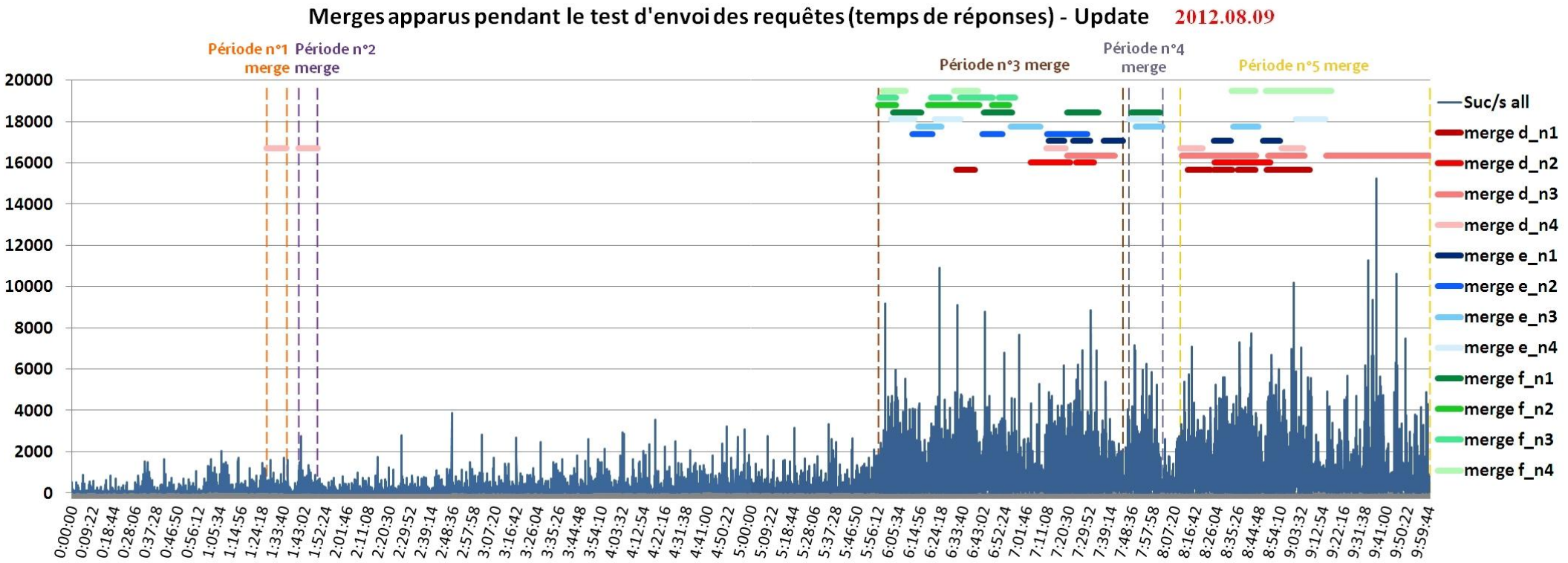
	Moyenne requêtes/seconde	Moyenne temps de réponse (ms)	Temps de réponse max (en ms) pour avoir une disponibilité de 99,99%
Maxi	53	28	5 500
Medium	53	26	5 540
Mini	54	25	5 490
Micro	49	52	5 580

- En update

	Moyenne requêtes/seconde	Moyenne temps de réponse (ms)	Temps de réponse max (en ms) pour avoir une disponibilité de 99,99%
Maxi	18	193	14 740
Medium	19	179	14 450
Mini	19	173	14 460
Micro	19	171	14 750

- Observations sur ce test :
  - Plus la charge augmente, plus les temps de réponses sont élevés mais dans un même « intervalle »
  - Les temps de réponse en update sont moins bons qu'en lecture
  - Apparition de merges dans la deuxième partie du test → augmentation temps de réponse et diminution requête envoyé

Test de Charge – Résultats (4/4)

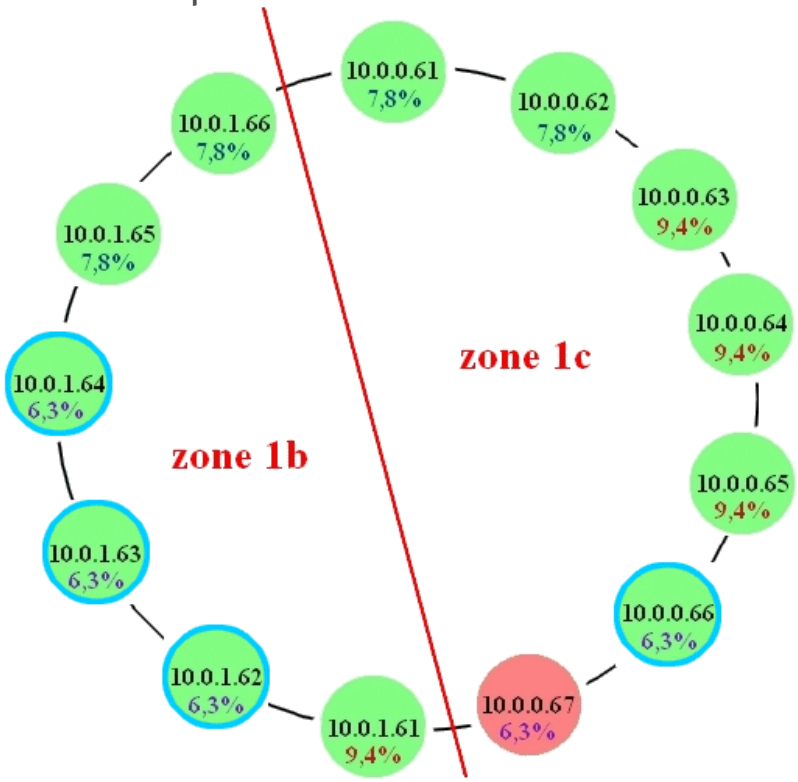


Test d'ajout d'un nouveau node au slave (1/3)

- **Pour ajouter un nouveau serveur au cluster, il faut procéder de la manière suivante avec Riak :**

  1- Sur un nouveau node déjà configuré et allumé, lancer la commande : riak-admin join *nom_node_1@adresse_node_1*

  2- Riak met en place de réorganisation de la répartition des données sur le cluster



  3- Le node 10.0.1.63 envoie environ 1,58% de ces données au nouveau nœud 10.0.0.67

Test d'ajout d'un nouveau node au slave (2/3)

4- Le node 10.0.1.62 envoie environ 1,58% de ces données au nouveau nœud 10.0.0.67
5- Le node 10.0.0.66 envoie environ 1,58% de ces données au nouveau nœud 10.0.0.67
6- Le node 10.0.1.64 envoie environ 1,58% de ces données au nouveau nœud 10.0.0.67

La procédure est complètement automatisée à partir du step 2 jusqu'au step 6.

Durée de l'opération :

Etapes importantes	Durée (consistency = 1)	Durée (consistency = 3)
Chargement des données venant de 10.0.1.63	15:25	14:37
Chargement des données venant de 10.0.1.62	16:30	17:17
Chargement des données venant de 10.0.0.66	17:04	18:04
Chargement des données venant de 10.0.1.64	17:11	16:35
Chargement des données sur le slave	**1:06:10**	**1:06:33**

Test d'ajout d'un nouveau node au slave (3/3)

- **Conclusion de ce test :**
  - La procédure d'ajout d'un node est très simple à exécuter, car très automatisée
  - Il y a des erreurs HTTP 503 jusqu'à ce que le nouveau node soit disponible
  - Les temps de réponses sont touchés par l'ajout d'un node :
    - Environ 6,5 millisecondes en plus pour les read
    - Environ 29,25 millisecondes en plus pour les write

Augmentation du facteur de réplication

- **Riak permet de modifier le facteur de réplication d'un bucket de la façon suivante :**
  - curl -v -X PUT -H "Content-Type: application/json" -d '{"props":{"n_val":5}}' http://10.0.0.61:8098/buckets/*nom_du_bucket*/props
- **Conclusion du test :**
  - Aucune erreur détectée, temps de réponse du changement de facteur de réplication très élevés (4 s)
  - Cette modification entraine de nombreux warning dans les logs qui nous indiquent que le temps d'exécution est long

Cette manipulation est très fortement déconseillée dans la documentation de Riak car les requêtes en cours considèrent que le facteur de réplication est toujours le facteur initial

Test de simulation de panne réseau

- **On envoie des requêtes d'update sur les nodes de la zone C et on simule une coupure réseau entre les nœuds de la zone C et de la zone B par « couple » de nodes (DROP IPTABLE).**
  - Exemple : 10.0.0.61 et 10.0.1.61
- **Conclusion du test :**
  - Très sensible aux pannes de réseau en consistance 1 (45% d'erreur 404 minimum)
  - Apparition d'erreurs 503 et 500 (>5/min) en consistance 3
  - Impacts sur les temps de réponses des reads et des updates

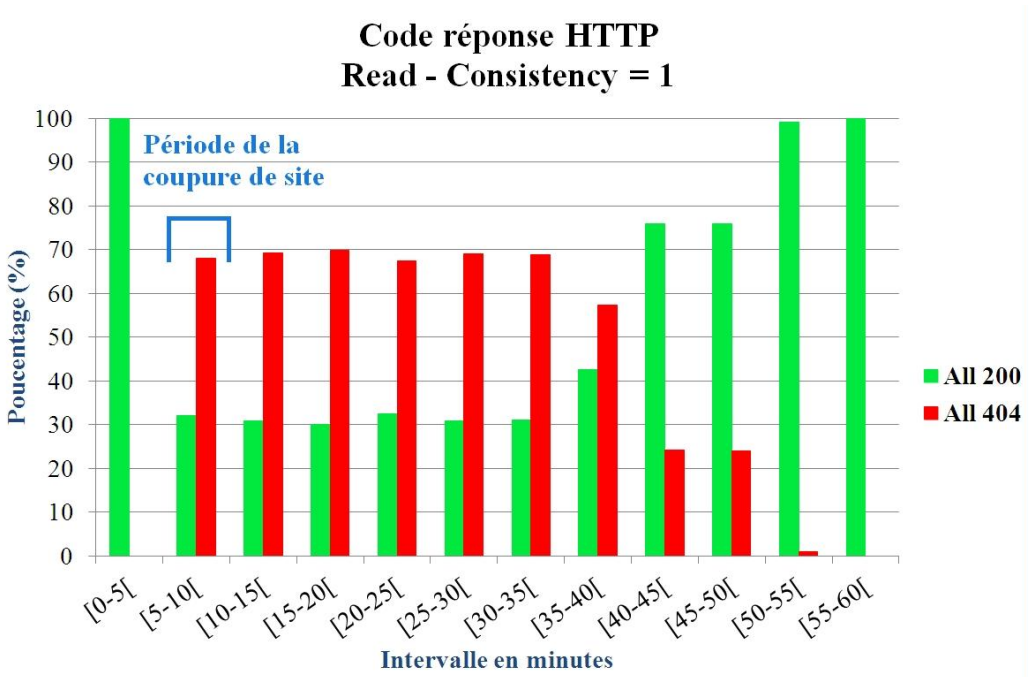
Riak a un comportement étrange en consistance 1

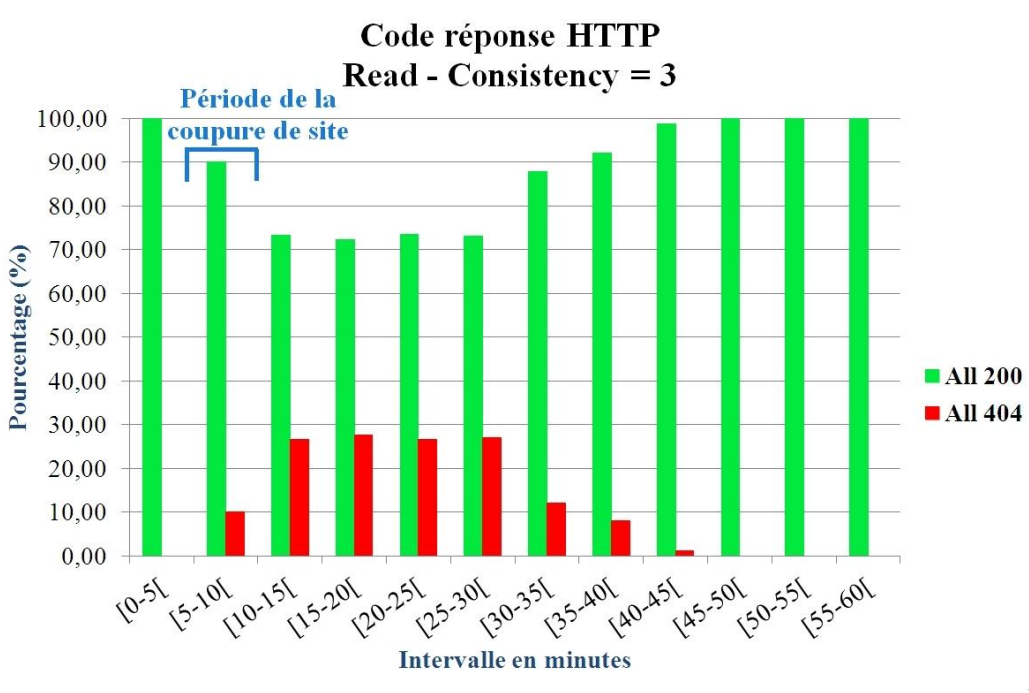
Période où aucune donnée n'est envoyé en consistance 3

Test de coupure de site (1/3)

- **Nous avons effectué une coupure de site en suivant la procédure suivant :**
  - Nodes de la zone B stoppé pendant 5 minutes avoir lancer l'envoi des requêtes
  - Après les 5 minutes de coupure, redémarrer les nodes
- **Résultats obtenus:**
  - Temps de réponses similaires pour les read en consistance 1 et 3
  - Temps de réponses plus élevés pour les write en consistance 1 (une dizaine de ms) et 3 (de 1 à 3 ms)
  - Plus d'erreurs en consistance 1 qu'en consistance 3 → utilisation d'un « basic-quorum » qui a un autre comportement que le quorum de base

Test de coupure de site (2/3)





Test de coupure de site (3/3)

- **Conclusion du test :**
  - Manifestation d'un comportement inattendu en consistance 1
  - Cela crée jusqu'à 70% d'erreurs en consistance 1 contre 28% en consistance 3
  - Rétablissement des requêtes au bout d'environ 46 min en consistance 1 contre environ 33 min en consistance 3 → ≈ 13 min de différence

Consistance 1 →bon temps de réponse mais en cas de coupure de site 70% des données non disponible en read

Test de backup/restore

- **La procédure de backup est très simple :**
  - Exécuter la commande suivante sur la machine sur laquelle on veut faire un backup :
    - riak-admin backup *nom_du_node cookie_du_node chemin_du_fichier_backup* [node|all]
  - Le backup est plutôt long (50 min en consistance 1, 42 en consistance 3)
  - Taille fichier backup 61 Go (taille dossier données 71 Go en consistance 1, 62 Go en consistance 3)
- **La procédure de restore est également très simple :**
  - Exécuter la commande suivante sur la machine sur laquelle on veut faire un restore :
    - riak-admin restore *nom_du_node cookie_du_node chemin_du_fichier_backup*
  - Le restore est toutefois très long (3h03 en consistance 1, 3h16 en consistance 3)
  - Taille du dossier contenant les données élevée (248 Go)
  - A noter : après le restore, une procédure de compression des données s'effectue automatiquement. A la fin, le dossier fait 62 Go

Test de crash de node

- **Lors de ce test, on arrête un node du cluster avec la commande *riak stop* alors que des requêtes d'écritures étaient envoyées par une application cliente.**
- **Avant le redémarrage de ce node, on efface toutes les données contenues dans le répertoire contenant les données et on redémarre le node**

- **Le node ne récupère pas les données perdues, mais seulement les valeurs des données qui ont été modifié après sa reconnexion**
- **Accès aux donnés n'entraîne pas d'erreurs**

Conclusions

- **Les +**
  - Installation et configuration faciles à mettre en place
  - De bons temps de réponses en lecture
  - Les procédures d'administration sont simples et bien automatisées
  - Possibilité d'utiliser du map/reduce pour faire des recherches ou des requêtes complexes (qui n'a pas été testé)
  - Possibilité de recréer un modèle avec relations entre objets
  - Possibilité de choisir le mode de stockage qui correspond le plus à ses besoins et possibilité de combiner plusieurs modes de stockage.

Conclusions

- **Les -**
  - Comportement étrange lors de la coupure entre 2 sites et de la simulation de panne en consistance 1
  - Lors du crash d'un node, non récupération des données
  - Restauration des données qui prend beaucoup de place avant la compression (dans les tests 248 Go)
  - Apparition des merges lors de la 2ème partie du test de charge qui a affecté les performances
  - On ne sait pas où sont placés les réplicas (il n'est pas garanti qu'ils soient tous localisés sur des nodes différents)
  - Quelques fois répartition non équitable des données entre les nodes

Riak dans le futur

- **Sortie de Riak 1.2 le 7 août 2012**
- **Cette version 1.2 a apporté de très intéressantes améliorations :**
  - L'administration du cluster a été changé :
    - Les commandes riak-admin join, leave et force-remove ont été dépréciées mais sont toujours utilisables
    - La nouvelle commande est riak-admin cluster *nom_commande*
      - *Il est possible d'utiliser les noms de commande suivants : join <node>, leave, leave <node>, force-remove <node>, replace <node1> <node2>, force-replace <node1> <node2>, plan, commit, clear*
    - Meilleure efficacité de l'ajout de plusieurs nodes au cluster en une fois
    - Possibilité d'ajouter des nodes au cluster pendant que l'on en retire
  - Meilleure visibilité des transferts actifs
  - Amélioration des parties Map/Reduce et statistiques

Merci ! ☺