Simulation d'algorithmes d'équilibrage de charge dans un environnement distribué

Identifications des besoins

Kevin Barreau Guillaume Marques Corentin Salingue

4 février 2015

Résumé

Dans une première partie, nous présentons le projet, le contexte et les hyothèses. Ensuite, nous développons les besoins fonctionnels et les besoins non-fonctionnels. De plus, nous dégageons une première version de la plannification du projet (GANTT). Enfin, nous présentons les livrables.

Sommaire

1 Définition du projet

1.1 Contexte

Définition Un environnement distribué est constitué de plusieurs machines (ordinateurs), appelées *noeuds*, sur lesquelles sont stockées des données.

L'expansion, au cours des deux dernières décennies, des réseaux et notamment d'Internet a engendré une importante création de données, massives par leur nombre et leur taille. Stocker ces informations sur un seul point de stockage (ordinateur par exemple) n'est bien sûr plus envisageable, que ce soit pour des raisons techniques ou pour des raisons de sûreté (pannes potentielles par exemple). Pour cela des systèmes de stockages dits distribués sont utilisés en pratique afin des les répartir sur différentes unités de stockages.

Définition Une donnée est un codage d'une information propre au système de base de données.

Définition Une requête est une interrogation d'une base de données afin de récupérer ou modifier de l'information sur des données.

Définition Une *charge* est associée à un noeud et désigne le nombre de requêtes que le noeud doit traiter.

Définition La réplication d'une donnée consiste à faire des copies de cette donnée sur d'autres noeuds.

Pour répartir toutes ces données, notre client a développé de nouveaux algorithmes d'équilibrage de charge basés sur la réplication qu'il souhaite tester dans un environnement distribué.

A bien placer :

Un noeud est une machine (ordinateur généralement) pouvant stocker des données et traiter des requêtes. Un noeud possède des données locales propres au fonctionnement du noeud.

1.2 Finalité

Nous devons développer une solution logicielle permettant de tester ces nouveaux algorithmes d'équilibrage de charge et de réplication dans un environnement distribué.

Il s'agira d'implémenter les algorithmes développés par le client. On distingue les algorithmes d'affectation de requête :

— **SLVO** Si la charge du noeud est inférieure ou égale à la charge minimum, il s'affecte toutes les requêtes en attente et en avertit les autres noeuds.
— **AverageDegree** Si la charge du noeud est inférieure ou égale à la charge moyenne, il s'affecte toutes les requêtes en attente et en avertit les autres noeuds.

Ainsi que l'algorithme de gestion de copie, permettant d'établir le nombre de réplicats d'un objet en fonction de sa popularité.

Pour comparer l'efficacité de ces algorithmes, on peut visualiser l'état du réseau *à tout moment*.

1.3 Hypothèses

Nous évoluerons dans un environnement distribué constitué de n noeuds de stockage dans lequel on souhaite stocker m objets. C'est un réseau statique, on ne peut pas ajouter ou supprimer de noeuds après création du réseau.

Données locales d'un noeud Un noeud contient les données locales suivantes :
— la charge de tous les noeuds du réseau
— la popularité de chaque objet stocké sur ce noeud
— une file d'attente de message à traiter
— la requête en cours de traitement

Requêtes Nous supposerons que les requêtes seront effectuées en un temps fixe.

2 Hiérarchisation des besoins

Nous avons dégagé des précédentes réunions, une liste de besoins fonctionnels et non-fonctionnels. Pour mieux les comparer, nous les avons hiérarchisés en fonction de leur priorité et de leur criticité.

2.1 Priorité

La priorité est un indicateur de l'ordre dans lequel nous devrons implémenter les besoins afin de satisfaire au mieux les exigences du client.

Valeur	Signification	Description
1	Priorité haute	A implémenter dans les premiers temps
2	Priorité moyenne	A implémenter
3	Priorité faible	A implémenter (en fonction du temps restant)

2.2 Criticité

Le niveau de criticité d'un besoin est un indicateur de l'impact qu'aura la non-implémentation de ce besoin sur le bon fonctionnement de l'application.

Valeur	Signification	Description
1	Criticité extrême	L'application ne fonctionnera pas
2	Criticité haute	Certaines fonctionnalités de l'application ne fonctionneront pas
3	Criticité moyenne	Certaines fonctionnalités seront perturbées
4	Criticité faible	L'application fonctionnera correctement

3 Besoins fonctionnels

3.1 Gestion d'un réseau

Un réseau est un ensemble de noeuds qui sont reliés entre eux (par exemple par Internet) et qui communiquent ensemble afin de traiter toutes les requêtes reçues.

3.1.1 Gestion des noeuds

Création d'un noeud (*Priorité* : 1, *criticité* : 1) Il est possible de séparer ce besoin en plusieurs sous-besoins :
— créer un noeud dans l'environnement
— initialiser les données locales d'un noeud

Mise à jour des données locales (*Priorité* : 1, *criticité* : 1) Afin de connaître l'état du réseau de manière précise, les données locales doivent être mise à jour à chaque action Une mise à jour a donc lieu lors du traitement d'un message dans la file d'attente.

Communication des données locales (*Priorité* : 1, *criticité* : 1) Un noeud doit être capable de communiquer ses données locales à d'autres noeuds du réseau.

Récupération de l'état du réseau (*Priorité* : 1, *criticité* : 1) L'application doit permettre la description de l'état du réseau. On souhaite connaître :
— le nombre de requêtes en attente
— la popularité des objets

3.1.2 Réplication d'un objet

Il s'agit de copier un objet sur un autre noeud.

Définition des fonctions de hashage (*Priorité* : 1, *criticité* : 3)

3.1.3 Popularité d'un objet

Les algorithmes à implémenter nécessitent de connaître la popularité d'un objet dans le réseau. La popularité d'un objet est fonction du nombre de requêtes sur cet objet. Plus ce nombre de requêtes est grand, plus l'objet est populaire.

Calcul de la popularité (*Priorité* : 1, *criticité* : 1)

Stockage de la popularité (*Priorité* : 1, *criticité* : 1) Chaque noeud stocke la popularité des objets qu'il contient.

Communication de la popularité (*Priorité* : 1, *criticité* : 1) Un noeud stockant des replicats doit communiquer la popularité de ces derniers au noeud possédant l'objet original.

3.2 Protocole de réaffectation

(*Priorité* : 1, *criticité* : 1) Les algorithmes d'équilibrage de charge à implémenter sont **SLVO** et **AverageDegree**.

3.2.1 Protocole de test

La conformité des algorithmes implémentés est assurée par un protocole de test suivant la démarche :
— Définir un réseau R, un ensemble d'objets O et un ensemble de requêtes Q
— Faire tourner l'algorithme à la main
— Stocker l'état final du réseau
— Faire valider ce processus par le client
— Exécuter l'algorithme avec R, O et Q
— Vérifier les résultats constatés avec les résultats attendus

S'il y a une différence entre les deux résultats, une vérification par le client peut être envisagée dans le cas de résultats *presque* similaires. La notion de similitude est laissée à l'appréciation de l'équipe en charge du projet, lors de la vérification.

3.3 Requêtes

3.3.1 Générations de requête

3.3.2 Importation d'un jeu de requête

Pour comparer l'efficacité des algorithmes, il doit être possible d'envoyer sur le réseau un même suite de requêtes, un jeu de requête.

Importation (*Priorité* : 1, *criticité* : 2) L'application doit pouvoir lire un fichier contenant une suite de requête et envoyer ces requêtes sur le réseau.

3.4 Visualisation des données

— Temps de réponse moyen sur les requêtes passées.
— Charge d'un noeud
— Popularité des objets

4 Besoins non fonctionnels

4.1 Cassandra

Cassandra est une base de données distribuée. Nous créons notre environnement de simulation à partir de la dernière version stable, Cassandra.

Le choix de cette solution nous a été fortement recommandé par le client. En effet, celui-ci dispose de connaissances sur cette application et pourra donc plus facilement intervenir s'il souhaite faire évoluer le projet en implémentant par exemple de nouveaux algorithmes.

4.2 Maintenabilité du code

Nous ne pensons pas que le projet sera totalement terminé le 8 Avril 2015, date de rendu du code et du mémoire. Pour cela, nous avons défini quelques normes pour que le projet puisse être repris. (a détailler)

4.3 Gestion d'un réseau

4.3.1 Communication entre noeuds

Algorithme Le calcul de la popularité nécessite l'implémentation de l'algorithme d'approximation Space-Saving Algorithm [ADA05].

Pour connaître l'état du réseau, il faut regrouper les données locales des noeuds. Nous cherchons donc a récupérer ces données en un temps raisonnable ($O(log(n))$ pour n noeuds).

Pour cela, nous nous appuierons sur le protocole `Gossip` [Fou14]. Périodiquement, chaque noeud choisi n noeuds aléatoirement dont un noeud *seed*, noeud en mesure d'avoir une connaissance globale du système, et il communique à ces noeuds ses statistiques (valeur de sa charge, objets les plus populaires...).

Ainsi, la connaissance globale du système se fait en $O(log(n))$.

4.3.2 Taille des données

La taille de chaque donnée est laissée à l'apréciation de l'équipe. Néanmoins, celle-ci doit être suffisamment importante, afin de permettre de créer des requêtes qui "stressent" le système pour avoir des résultats cohérents (sur la base de l'hypothèse : chaque requête prend un même temps à être traitée).

4.4 Visualisation des données

Une vue correspond à une fenêtre de l'application, c'est à dire ce que voit l'utilisateur.

4.4.1 Etat du réseau

La vue permet de montrer l'état du réseau.

Le réseau est représenté par un graphe, les machines par des noeuds. Pour chaque machine, les données affichées sont la charge ainsi que le contenu de la file d'attente.

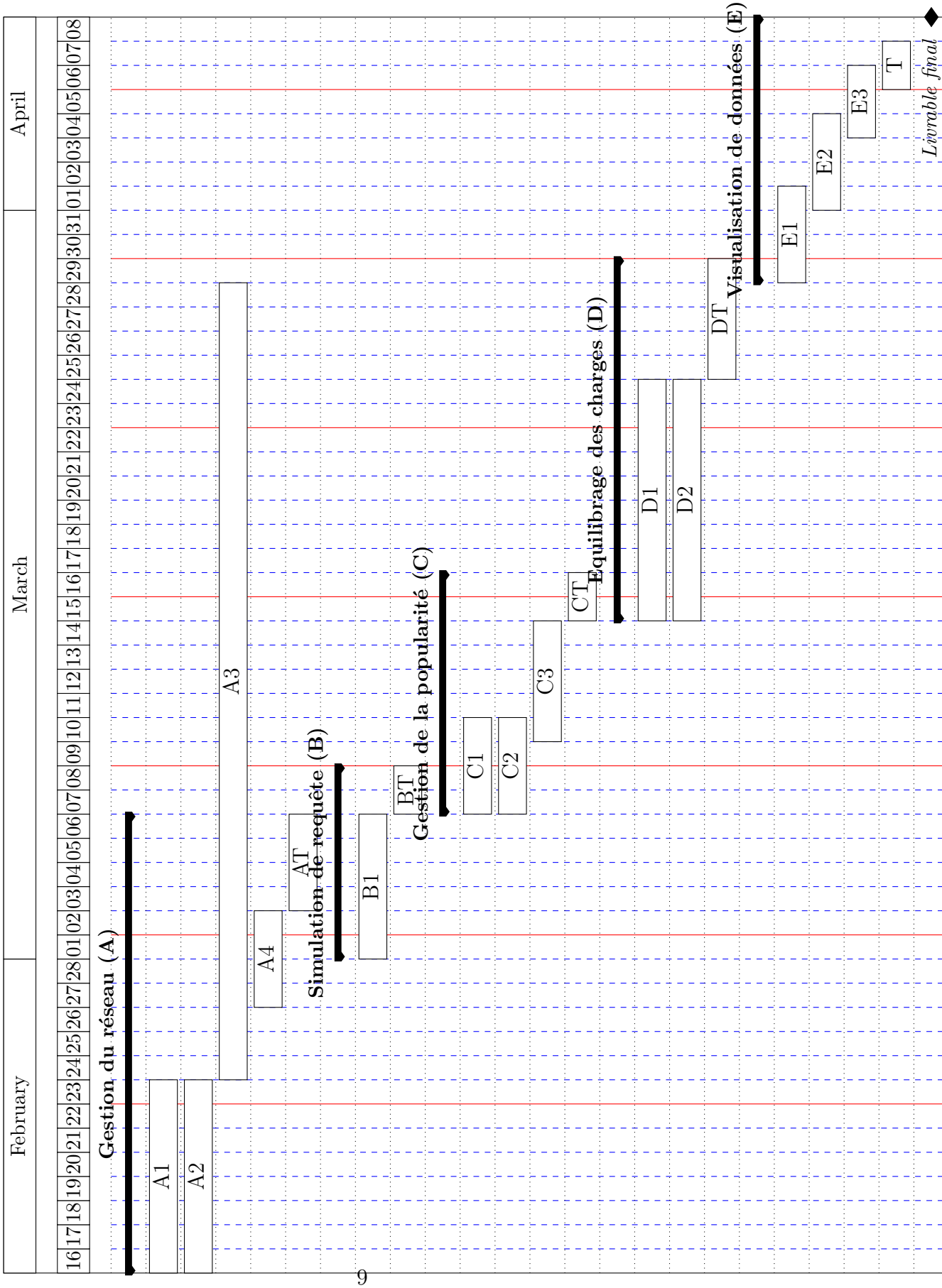
4.4.2 Actualisation de la vue

L'état du réseau doit-être visible en temps réel.

La vue peut donc être actualisée toutes les 0.5 secondes. Un délai plus faible risquerait de la rendre invisible (données clignotantes sur l'écran).

5 Répartitions des tâches

5.1 Diagramme de Gantt



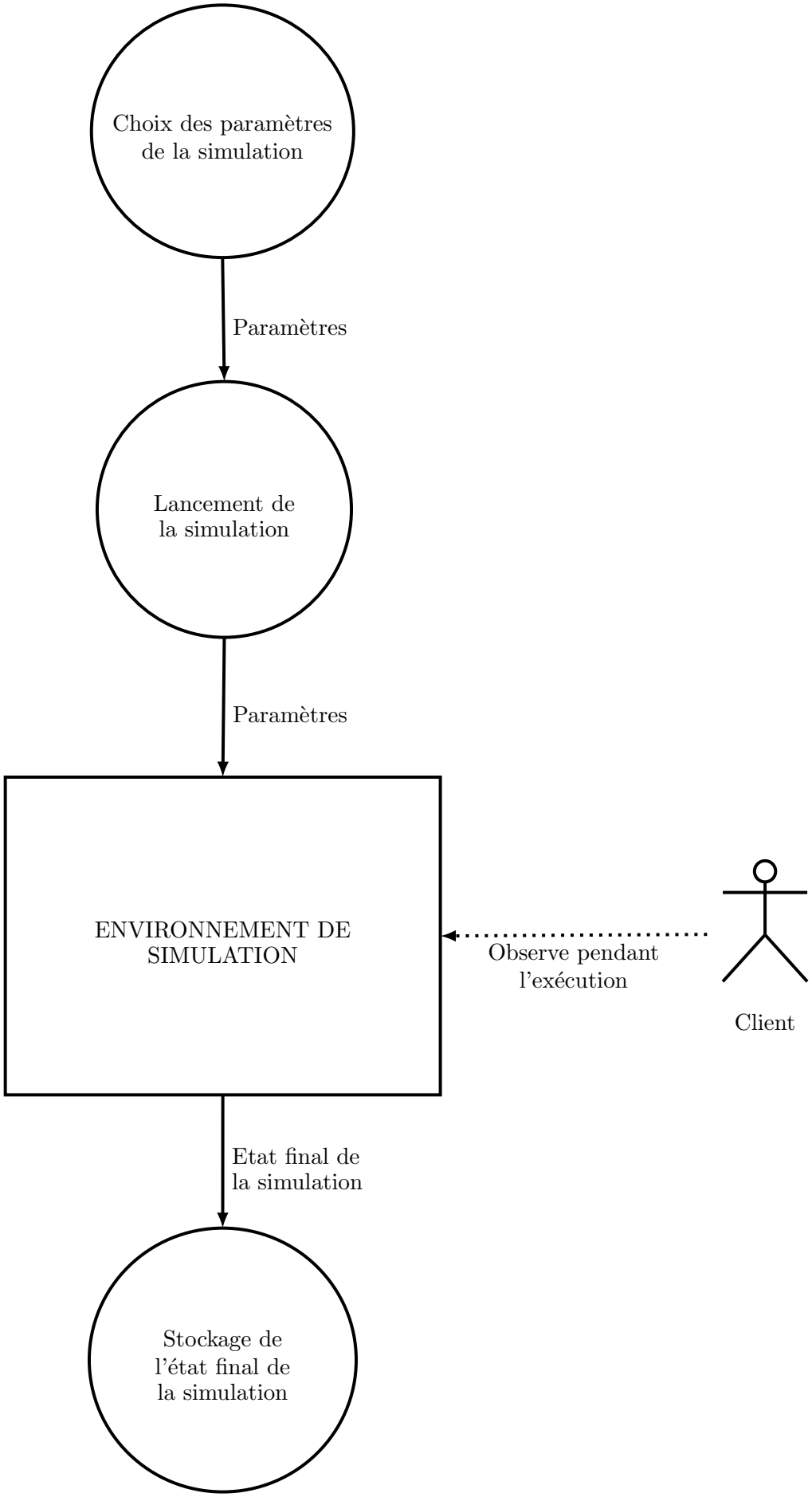
5.2 Affectation des tâches

Fct	Description	Développeur(s)	Commentaire
A1	Création des noeuds		
A2	Données locales des noeuds		Initialisation et implémentation
A3	Communication entre noeuds		
A4	Gestion des replicats		
AT	Tests groupe A		Vérification, tests, mémoire
B1	Générateur de requêtes		A détailler
BT	Tests groupe B		Vérification, tests, mémoire
C1	Popularité objet sur noeud		
C2	Space-Saving Algorithm		
C3	Popularité d'un objet		
CT	Tests groupe C		Vérification, tests, mémoire
D1	Implémentation SLVO		
D2	Implémentation AverageDegree		
DT	Tests groupe D		Avec client
E1	Prise en main Tulip		
E2	Représentation réseau		
E3	Représentation données		
T	Tests finaux		Vérification, tests, mémoire

6 Livrables

6.1 Livrable “final”

Il devra être remis le 8 Avril 2015.


Choix des paramètres de la simulation
Paramètres
Lancement de la simulation
Paramètres
ENVIRONNEMENT DE SIMULATION
Observe pendant l'exécution
Client
Etat final de la simulation
Stockage de l'état final de la simulation

Références

[ADA05] Metwally A, Agrawal D, and El Abbadi A. Efficient computation of frequent and top-k elements in data streams. 2005.

[Fou14] The Apache Software Foundation. Architecturegossip - cassandra wiki. <`http://wiki.apache.org/cassandra/ArchitectureGossip`>, 2014. [Accessed 21 January 2015].