Etat de l'art pour Pastry

Anton CLAES

16/03/2017

Contents

1 Articles

1.1 Bigtable: A Distributed Storage System for Structured Data [2]

Bigtable est un middleware de chez google pour le stockage et le travail sur des données distribuées. Bigtable est utilisé dans plus de 60 projets google. Il permet de travailler sur une grand diversité de charges de travail et de stockage. Bigtable a beaucoup de concepts calqués sur ceux d'une base de données. Bigtable ne supporte pour autant pas un modèle de stockage relationnel.

Cet article décrit principalement le fonctionnement de Bigtable en tant que système distribué de stockage, générique et adapté à plusieurs applications. Il met l'accent sur le rangement des données, leur traitement et la manière dont

un client y accède. Cela repose principalement sur des tables et tablettes, qui sont des unités de stockages logiques, implémentées au dessus du système de fichier distribué de Google : le Google File System (GFS).

Chaque entrée est identifiée dans la table par une clé de ligne et de colone, ainsi qu'un marqueur temporel (timestamp). Puis, les entrées dont les clés de colonnes sont proches sont regroupées en tablettes.

Ces tablettes sont ensuite dupliquées 5 fois, sur 5 serveurs de tablettes différents (Système Chubby). La consistence des données est garantie, par l'algorithme de Paxos. Une donnée est considérée active si la majorité de ces tablettes sont actives. Un seul serveur de tablettes délivre l'information à la fois, les autres serveurs ne délivrent la tablette que si le principal est tombé.

Bigtable gère aussi un système de compression de données, dont l'algorithme est choisi par le client de bigtable. Dans tous les cas la compression est axée sur une performance temporelle plutot que spatiale.

1.2 Dynamo: Amazon's Highly Available Key-value Store [4]

Dynamo est le système de gestion de stockage de données d'Amazon. Il est capable de traiter des données de natures très différentes. Il assure un état persistant et de la fiabilité aux données, pour répondre aux contraintes de la vente.

Les pannes sont considérées comme un comportement normal dans d'aussi grands réseaux distribués. La tolérance aux pannes se fait grâce à deux mécanismes :

Chaque donnée stockée est dupliquée sur un certain nombre de noeuds, et elle n'est ecrite et lu que depuis les premiers noeuds de l'ensemble à laquelle elle a été assignée. Puis, un second mécanisme permet de synchroniser les données pour maintenir leur cohérence.

L'architecture logicielle se compose d'un service de coordination de requêtes, d'un service de maintient des members du réseaux et de détection de pannes et d'un service de stockage local de données. Ces services communiquent ensemble et peuvent être interfacés avec d'autres technologies (base de données, systèmes de fichiers...)

La performance de Dynamo se fait au prix de compromis sur la cohérence des données, leur disponibilité, leur rentabilité. Les bases de données relationnelles ne permettent pas d'atteindre une performance suffisante, c'est pourquoi Dynamo utilise un versionnage des données et un mécanisme de résolution de conflits pour palier aux éventuelles incohérences.

La topologie du réseau peer to peer est basé sur un anneau mais chaque noeud ne se voit pas assigné un mais plusieurs voisins. De plus un seul serveurs physique se voit souvent attribuer plusieurs noeuds virtuels pour permettre l'équilibrage des charges au sein du réseau.

Le passage à l'echelle se fait de manière dite incrémentale : un noeud est ajouté à la fois. Le système supporte des serveurs et des noeuds avec des ca-

pacités différentes et adapte leurs charges en conséquence. Certains noeuds du réseau ont un role de coordination.

1.3 New Algorithms for Load Balancing in Peer-to-Peer Systems[6]

Cet article présente deux méthode d'équilibrage de charges pour les réseaux peer to peer, en prenant l'exemple de Chord, un réseau p2p classique.

L'équilibrage est un problème majeur pour la performance des réseaux p2p. L'approche classique consiste à adopter une topologie d'anneau, et d'établir des voisins qui comuniquent entre eux, en utilisant des addresses. Le problème réside dans la manière d'attribuer les addresses : c'est le plus souvent des Distributed Hash Tables (DHT) qui sont utilisées. L'attribution des addresses ainsi que l'attributions de données (items) aux noeuds se fait par Hash, de manière aléatoire.

Le problème soulevé par l'approche distribuée des tables de hashage est que la distribution des items sur les noeuds est basée sur le hash d'une clé d'item. Or, si la distribution des clés d'entrées n'est pas homogène alors la distribution des hash ne le sera pas non plus et certains noeuds seront surchargés.

Une technique pour résoudre ce problème consiste à créer des noeuds virtuels, mais cela a des inconvénients en terme de charge réseau, cela crée un surcout protocolaire important.

Il faut donc envisager une modification de l'attribution des hash, et permettre d'exploiter les plages de hash faiblement attribuées. Il faut cependant le faire sans permettre aux noeuds de s'attribuer des plages de hash pour éviter qu'un noeud pirate ne viennent rendre le système défaillant.

La seconde approche consiste à effectuer une répartition de charge en déplaçant des items depuis des noeuds surchargés vers des noeuds sous-chargés.

1.4 Stable distributed P2P protocols based on random peer sampling [8]

Cet article décrit trois protocoles différents de partage de données dans un réseau peer-to-peer. Le problème abordé est celui de la mauvaise distribution de morceaux de données, qui aboutissent à beaucoup de clients p2p ayant juste un morceau manquant. C'est particulièrement le cas si la séléction de morceaux se fait de manière aléatoire.

Les protocoles prennent en compte le fait qu'un noeud puisse ne pas être altruiste et quitter le réseau dès la fin de son téléchargement. Il faut mettre en place un stratégie de sélection des morceaux.

Le problème de l'étude réside aussi dans le mécanisme de simulation, et ses hypothèses associées. Par exemple ici les problème liés au réseau (IP) ne sont pas évoqués.

Les protocoles étudiés sont donc les suivants : (k nombre de morceaux du fichier)

Common Chunk Protocol : Téléchargement des morceaux les plus rares depuis 3 pairs - Téléchargement d'un morceau parmis les ceux qui lui manquent, sans rareté - Téléchargement du dernier morceau depuis 3 pairs si le morceau est présent sur 2 des 3 pairs. Le protocole est stable.

Rare Chunk Protocol : Un pair connecté ne télécharge un morceau de fichier que si celui si est rare. Sinon il ne télécharge rien. The protocole est stable.

Rare Chunk Protocol (extended): Ce protocol est un protocole basé sur Rare chunk selection qui prend en compte le fait qu'un pair puisse mentir sur les données qu'il possède afin de ne pas les partager. Avec ce protocole, ne pas mentir maximise le débit. Si deux pairs ont des morceaux rares différents, ils l'échangent. Si un seul en a un, alors il ne sera partagé qu'avec une probabilité 1-p, et si aucun n'en a, alors il n'y a pas d'échange.

1.5 Enabling Robust and Efficient Distributed Computation in Dynamic Peer-to-Peer Networks [1]

Cet article donne un protocol peer to peer entièrement distribué qui possède des propriétés interessantes d'expansion du réseau et de réorganisation, tout en étant simple et léger pour les machines. En particulier, il gère bien le départ et l'arrivée constant de noeuds. Certaines hypothèses sont faites, par exemple le nombre de pairs dans le réseau est supposé stable, même s'ils changent.

Le protocole permet une organisation aléatoire robuste avec un système de maintenance qui permet de réorganiser le réseau lorsque trop de connections sont tombées.

Le principe d'organisation est le suivant : chaque noeud génère des jetons (tokens) qui contiennent les identifiants (ID) de la source dans le réseau et chaque noeud transmet le token aléatoirement dans le réseau, un certain nombre de fois (borné). Puis une fois que le jetons a fait un nombre suffisants de sauts aléatoire (random walk), il est considéré comme mature, et définit ainsi les voisins de la source dans le réseau.

Cette phase initiale du réseau est dite "Bootstrap", pendant cette phase l'expansion du réseau est relativement faible, puis le réseau passe en phase normale. Pendant cette phase, le protocole se prépare pour un évolution future plus virulent du réseau.

Lors de sa connection, un noeud va d'abord chercher à récupérer des tokens matures en envoyant des tokens à ses voisins, (phase bootstrap) puis quand c'est fait il va avoir un jeu de tokens valides avec lesquels échanger des données.

Si le nombre de tokens tombe sous un seuil, il les jette tous et entre en mode de reconnection, dans lequel il va chercher à recréer des tokens neufs.

Lorsqu'un noeud met en place un lien avec un voisin, il le fait en accord avec celui-ci : "two-step handshake". Chaque noeud peut demander une connection avec un autre noeud. Celui-ci peut l'accepter ou la refuser expressément, ou alors ne pas répondre. Un noeud peut faire tomber un lien sans prévenir l'autre noeud.

1.6 An efficient hypercube labeling schema for dynamic Peer-to-Peer networks [9]

Cet article montre un protocole de gestion d'une topologie peer to peer optimisée grâce aux principes d'hypercubes : HyperD. Il est fait pour diminuer le nombre de sauts entre noeuds du réseau, et ainsi diminuer l'overhead et la bande passante consommée.

Chaque noeud ce voit attribuer un ou plusieurs labels, qui sont des champs de bits définissant leur position dans un hypercube. La dimension de l'hypercube correspond au nombre de bits du label. Il en découle beaucoup de propriétés de calculs interessantes, par exemple deux noeuds sont voisins si un seul bit d'un seul de leurs labels diffère. La distance entre deux noeuds est le minimum des distances de Hamming entre leurs labels. Cela permet de faire des calculs très rapides de distance et donc de chercher la plus courte distance.

Les performances sont ainsi bien meilleures que les réseaux construits de manière aléatoire.

Le routage se fait de la manière suivante : si le noeud de destination est un voisin, alors on transmet aux voisin, sinon on transmet à un de ses voisins qui est le plus proche de la destination en distance de Hamming.

La dimension de l'hypercube dépend du nombre de noeuds dans le réseau, c'est la plus petite puissance de deux supérieure au nombre de noeuds. Par propriété de l'hypercube le réseau est facilement extensible, car un hypercube de dimension N est compris dans un hypercube de dimension N+1.

L'ajout d'un noeud se fait en se connectant à un noeud et en faisant un requête de connection. Cette requete est broadcastée par tous les noeuds qui n'ont pas de label disponible, sinon ils répondent et le nouveau noeud prend un label parmis ceux retournés.

Un noeud peut quitter le réseau en prévenant ou tomber en panne. Si il prévient ses labels sont réassignés à ses plus proches voisins. Une perte d'un noeud n'affecte donc que ses voisins les plus proches

1.7 A Simpler Load-Balancing Algorithm for Range-Partitioned Data in Peer-to-Peer Systems [3]

Cet article présente un algorithme d'équilibrage des charges dans une réseau pair à pair, afin de garantir un rapport des différences de charges des noeuds bornés, et ce même quand des noeuds se voient attribuer ou supprimer de nouveaux éléments.

Cet algorithme est basé sur l'algorithm AdjustLoad, qui se sert de deux opération basiques pour faire l'équilibrage des charges : NbrAdjust et Reorder, qui permettent respectivement de partager la charge avec un noeud voisin et de se déplacer dans le réseau pour partager la charge d'un noeud.

Cependant cet algorithme, récursif, est difficile à implémenter, et c'est pourquoi il a été simplifié. L'algorithme simplifié possède aussi deux opérations d'équilibrage : MinBalance et Split.

Minbalance est appellée quand il y a un ajout d'un élément sur un noeud, au dessus d'un seuil d'éléments. Dans ce cas, le noeud le moins chargé du réseau transmet sa charge a son voisin le moins chargé puis reçoit la moitié de la charge du noeud le plus chargé.

Split est appellée quand un noeud se voit retirer une charge en dessous d'un seuil d'éléments. Dans ce cas, soit il absorbe une partie de la charge de ses voisins, soit il transfère toute sa charge à ses voisins et prend la moitié de la charge du noeud le plus chargé.

Ce transfert de charge a un cout : il faut transférér la donnée si elle bouge, mais aussi propager les changements de localisation de données des noeuds après transfert. Il faut aussi être capable d'évaluer les charges minimales et maximales dans le réseau, ainsi que des informations globales, ce qui est couteux.

1.8 Efficient Range Query Processing in Peer-to-Peer Systems [7]

Cet article présente Armada, un système de recherche de données dans un réseau peer to peer capable d'effectuer des requêtes sur un plage de valeurs, avec garanties de délai. Armada supporte l'ajout et la recherche d'objet selon des plages d'attributs simple ou multiples. Armada ne se charge pas de gérer la topologie du réseau, c'est fait par FissionE sur lequel Armada est construit.

Le système est basé sur des algorithmes de hashages qui conservent au moins partiellement l'ordre après hashage. Puis il construit un arbre de routage pour effectuer la correspondance avec la topologie FissionE. Les infrastructures DHT classique ne supportent pas les requêtes par plages mais seulement par cle exacte, pourtant certaines applications requièrent des requêtes sur des plages de valeurs, ce qui peut être très lent.

Deux arbres sont utilisés :

L'arbre de partition permet de faire correspondre des objets d'IDs proches à un même noeud ou a des noeuds en lien. L'arbre de routage (Forward Routing tree - FRT) permet d'établir la correspondance entre les requêtes et la topologie réseau sous-jacente.

L'algorithme de routage de FissionE se base sur un nommage des noeuds dans un espace de Kautz, et sur un routage par graphes de Kautz. Cela permet de garantir des requêtes de compléxités bornées par 2logN, et un cout protocolaire lors de l'ajout d'un noeud borné par 3logN.

Les objets peuvent être séparés en deux catégories : identifiés par un clé unique ou par un ensemble d'attributs. Dans le premier cas, Armada utilise un nommage et une recherche par nommage simple, grâce à l'algorithme de Simple Hash, qui conserve l'ordre des attributs. C'est aussi compatible avec la tolérance aux pannes et l'équilibrage des charges au sein du réseau.

Dans le second cas, un objet est caractérisé par un tuple d'attributs qui le caractérisent. Dans ce cas l'algorithme de Mutliple Hash est utilisé, celui ci ne permet qu'une conservation partielle de l'ordre des attributs après Hashage. Cet inconvénient est contré par l'algorithme de MIRA, qui permet de déterminer si un noeud possède la donnée avant de lui envoyer une requête.

1.9 Heterogeneity and Load Balance in Distributed Hash Tables [5]

Cet article présente le middleware Y0, qui est une optimisation de Chord permettant un équilibrage des charges moins couteux que les autres solutions existantes. Y0 est non seulement capable de gérer des capacités de noeuds différentes au sein du réseau mais il se sert de cette propriété des réseaux peer to peer pour optimiser leur routage.

Les solutions classiques utilisant une table de hashage distribuée imposent de maintenir une importante table de routage et de contacter beaucoup de pairs pour effectuer une requete. L'équilibrage des charges s'y fait classiquement en créant plusieurs noeuds virtuels sur une machine physique, mais cela implique d'avoir autant de table de routage que noeuds virtuels, le surcout est globalement important.

Pour pallier à ce problème, Y0 utilise également plusieurs noeuds virtuels, mais leurs IDs sont choisis proches et ils peuvent donc partager la même table de routage, ce qui évite ce surcout.

D'autre part, le réseau s'organise pour permettre de faire de l'hétérogénéité des noeuds un avantage et pas un inconvenient. Pour ce faire, les noeuds les plus puissant selon une métrique donnée (CPU, mémoire, bande passante...) sont assignés au routage et les noeuds qui sont en dessous d'un certains seuil de puissance relativement aux autres noeuds n'effectuent pas de routage. Ainsi les routes sont connues et passent toujours par les mêmes noeuds, elles sont donc plus courtes, ce qui limite la congestion. Les noeuds doivent déterminer leur puissance relativement aux autres noeuds du réseau, ils peuvent par exemple le faire en la comparant à celle de leurs voisins ou de noeuds aléatoires. De plus ce placement de noeuds permet un meilleur controle de la topologie du réseau.

Y0 garantit que chaque noeud ne se vera pas assigner plus de 3.6 fois la charge qui lui est équitablement due.

References

[1] J. Augustine, G. Pandurangan, P. Robinson, S. Roche, and E. Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 350–369, Oct 2015.

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[3] Jakarin Chawachat and Jittat Fakcharoenphol. A simpler load-balancing algorithm for range-partitioned data in peer-to-peer systems. *Netw.*, 66(3):235–249, October 2015.

[4] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[5] P. B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 1, pages 596–606 vol. 1, March 2005.

[6] David Karger and Matthias Ruhl. New algorithms for load balancing in peer-to-peer systems. 2003.

[7] D. Li, J. Cao, X. Lu, and K. C. C. Chen. Efficient range query processing in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):78–91, Jan 2009.

[8] B. Oğuz, V. Anantharam, and I. Norros. Stable distributed p2p protocols based on random peer sampling. *IEEE/ACM Transactions on Networking*, 23(5):1444–1456, Oct 2015.

[9] Andi Toce, Abbe Mowshowitz, Akira Kawaguchi, Paul Stone, Patrick Dantressangle, and Graham Bent. An efficient hypercube labeling schema for dynamic peer-to-peer networks. *J. Parallel Distrib. Comput.*, 102(C):186–198, April 2017.

2 Discussion

2.1 Comparaison

Article	Adaptabilité face aux pannes muettes	Adaptabilité face aux départs/arrivées	Adaptabilité face aux capacités différentes	Organisation automatique	Polyvalence applicative	Complexitée bornée	résistance aux noeuds malveillants	Versionnage des données	Equilibrage des charges
Bigtable[2]	+	+/-	N/A	- -	++	N/A	-	++	N/A
Dynamo[4]	++	+	+	++	+	+	- - -	++	N/A
New Algo... [6]	N/A	N/A	++	+	+	N/A	++	- -	+++
Stable p2p[8]	+++	+++	++	++	-	-	+++	- -	+
Robust efficient p2p[1]	+++	+++	N/A	+++	N/A	-	N/A	- - -	-
Hypercube[9]	++	+++	N/A	+++	+	+++	N/A	- - -	N/A
Load balancing algo[3]	++	+	+++	++	+	++	N/A	- - -	+++
Range query p2p[7]	N/A	N/A	N/A	++	++	+++	N/A	- - -	++
Heterogeneity, Load balance[5]	++	++	+++	++	+	+++	N/A	- - -	+++

2.2 Critique pour l'IoT

1. [2] Bigtable est un système dédié au stockage. Sa polyvalence de stockage et sa tolérance aux pannes sont interessantes pour l'IoT mais son principal défaut est d'être malgré tout seulement partiellement décentralisé. En effet un serveur effectue une répartition des données et de l'équilibrage de charges.

2. [4] Comme bigtable, Dynamo est très polyvalent en termes de données stockées comme de méthodes d'accès. Il est parfaitement tolérant aux pannes, et permet de garantir la cohérence des données. Il est capable de s'auto-organiser, mais seulement de façon incrémentale. Son adaptation aux réseaux peer to peer IoT n'est donc pas simple.

3. [6]Le système présenté permet l'équilibrage des charges au sein du réseau

peer to peer, ce qui est une propriété nécéssaire dans les réseaux avec des capacités différentes comme l'IoT. Toutefois l'article permet plus d'apporter des solutions aux déséquilibres induits par la topologie que par les capacités différentes des noeuds.

4. [8]Cet article donne des solutions aux problèmes de données partiellement manquantes, ainsi qu'au fait qu'un noeud ne pas être honnête et chercher à profiter du réseau sans y contribuer. Il propose en particulier un protocole qui rend le comportement honnête le plus interressant. Le concept est interessant pour l'IoT car il est évident qu'un noeud pirate peut être amené à se connecter à un réseau IoT.

5. [1]Cet propose une solution d'organisation du réseau peer to peer entièrement distribuée et capable de supporter le départ et l'arrivée constant de noeuds. C'est extrêmement interessant pour l'IoT car les noeuds sont succeptibles de tomber en panne ou de quitter et rejoindre le réseau de manière très importante, par exemple s'ils cherchent à économiser de l'énergie. Il faut que le réseau soit robuste malgré cela.

6. [9]L'article décrit une topologie de réseau peer to peer très interessante car elle permet de faire un routage efficace tout en permettant un passage à l'echelle efficace. C'est très interessant pour l'IoT car les noeuds ont souvent des capacités faibles et il faut donc limiter la puissance de calcul utilisée, tout en ayant un routage efficace. Le protocole proposé determine la route la plus courte grâce à des calculs de distances de hamming, peu couteux.

7. [3]Cet article propose une solution au problème d'équilibrage des charges et garantit une différence de charge bornée entre les noeuds les plus chargés et les moins chargés. C'est très interessant pour l'IoT car les noeuds surchargés tomberont vite. Toutefois l'équilibrage des charges a un cout car il se fait par déplament de données et qu'il faut une information globale sur le réseau.

8. [7]Armada propose un stockage des données capable de rechercher des objets par plages de clés, il évite de brasser toutes les données et conserve un ordre dans les clés au moment du hashage. Dans le domaine de l'IoT cela permettrait de gérer la localisation des données ou encore de faire efficacement des requetes sur des noeuds qui sont en lien les uns avec les autres. Toutefois Armada repose sur la topologie peer to peer FissionE, efficace mais dont les caractéristiques nécéssite étude avant d'être utilisées pour l'IoT.

9. [5]Y0 est un optimisation du célèbre Chord. Il permet un équilibrage des charges moins couteux que Chord et un routage efficace profitant des capacités variables des noeuds. Cette dernière propriété est très interessante dans l'IoT car les noeuds sont très sujets à voir des capacités différentes, le fait de pouvoir en tirer profit ne peut donc pas être négligé.