Un langage qui révolutionne l'IoT et les architectures many-core : le langage Go, retours d'expériences

École Polytechnique- Palaiseau

Jeudi 30 juin 2016

Coordination Scientifique

Michel Batto (INRA)

Editorial Board

Un langage qui révolutionne l'IoT et les architectures many-core : le langage Go, retours d'expériences

Séminaire Aristote, 30/06/2016 à l'École Polytechnique

Coordination scientifique

Michel Batto (INRA)

Partenaires

Table des matières

Compte-rendu des interventions

Introduction

80 personnes participent à la journée introduite par Jean-Michel Batto, responsable informatique à l'unité Micalis de l'Inra (Institut national de la recherche agronomique). Lui qui a monté le séminaire commence en présentant l'association Aristote qui existe depuis les années 1980 pour réfléchir sur les pratiques numériques et favoriser les échanges et les expériences de personnes d'horizons différents. Ce séminaire a été difficile à monter car il y a peu d'intervenants. Le langage Go (aussi appelé Golang) a émergé en 2009. En analysant le web, Jean-Michel Batto s'est rendu compte que ce langage est aujourd'hui au même niveau que le langage C ou que Javascript. C'est un langage opensource de type BSD et compilé. Jean-Michel Batto indique que le séminaire aurait pu être orienté vers la formation mais le retour d'expériences est davantage dans le style d'Aristote.

Le 17 janvier dernier, IBM a porté le langage Go sur system/Z. Go est présent dans l'infrastructure et les applications Cloud. Au 30 juin 2016, sur GitHub il y a 14,6 millions de *repositories*, dont 78% ont été créées depuis le 1er janvier 2015. 10% des *repositories* ont plus d'une étoile (qualité), 0,05% en ont plus de 1000. Ceux de plus grande qualité sont en Javascript et en Java, mais Go apparait de plus en plus dans cette liste.

1. Un nouveau langage, pourquoi faire ?

Thierry Goubier (CEA)

Thierry Goubier, ingénieur de recherche au CEA, a travaillé sur l'évolution des codes. Aujourd'hui sur Go, un des aspects intéressants est de savoir pourquoi certaines choses fonctionnent ou pas. Pourquoi ces nouveaux langages apparaissent et résistent dans ce domaine où il y en a beaucoup.

Quand on crée un langage, on veut dominer le monde et résoudre tous les problèmes en informatique. Cela peut être aussi à la suite d'une rupture technologique. OS (Unix) était portable, adaptable avec un compilateur simple et performant. Aujourd'hui les langages de programmation ont évolué. Mais dans quel sens ? Bon ou mauvais ? Ne faut-il pas un nouveau langage pour dominer le monde ?

Le langage Go est improbable, car C a évolué en C++ qui intègre tout et sait tout faire. qui domine l'industrie, mais certains informaticiens programment encore en C, en fortran, en Python, en Java etc. C++ n'a pas rempli tous les usages. Il n'est donc pas le langage idéal. C'est normal, car un langage est un compromis. Il est utilisé dans un contexte qui change avec le temps (processeur) et on a fait des progrès en génie logiciel.
La création d'un nouveau langage de programmation n'est pas seulement la liste de ce qu'on veut en faire, mais aussi de ce qu'on ne veut pas en faire.

Depuis C, les logiciels sont plus complexes, on a introduit des modules, des objets, des types de données abstraites. On a introduit aussi des concepts qui rendent les architectures plus extensibles et qui permettent de structurer ces architectures.
Quand on conçoit un langage, on veut l'isoler du matériel, qu'il nous donne des outils pour exprimer des algorithmes, maîtriser la complexité, exploiter la puissance du

matériel. Un langage conditionne la représentation que se fait le programmeur de la machine. Il rend facile certaines formes, structures ou manières de travailler et rend difficile les formes que le concepteur ne souhaite pas. Ce conditionnement du programmeur n'est pas toujours conscient.

Thierry Goubier revient à Go en notant que ses pères sont 3 personnes (Griesemer, Pike, Thompson) étaient déjà impliqués dans d'autres langages. Tout ce que Go propose et contient, existe déjà dans d'autres langages, mais son pouvoir est dans le choix des fonctions et des concepts qu'il intègre et dans la manière dont elles sont intégrées. C'est ce que l'histoire a dicté. Par exemple, les fermetures ont été définies dans Scheme en 75 mais Java ou C++ ne les ont pas intégrés au début.

Les choix de Go sont d'être un langage proche de C, une modularité par composition, des goroutines pour le parallélisme multi-cœur et many-cœur ainsi que des interfaces pour rendre les types extensibles. Les cas simples sont statiques, les cas complexes sont résolus à l'exécution. De plus, la mémoire est gérée automatiquement. Il n'y a pas d'exceptions contrairement aux autres langages mais des retours d'erreurs. Il comprend des outils avancés comme le compilateur, le parseur, la réflexion. La compilation est rapide et la syntaxe est simplifiée.
L'intérêt de Go est sa simplicité à programmer et l'aisance à maîtriser. C'est un langage plus proche des langages des années 80 avec tous les bénéfices de la recherche (*closure*, composition, csp, méta-programmation).

Thierry Goubier note quelques exemples des nouveautés du langage Go: la composition en lieu et place de l'héritage, la simplification du retour des résultats et l'association résultat-erreur. Comme il n'y a pas d'exception, la fonction *panic* indique que le programme explose. À noter aussi, les fonctions anonymes, les goroutines pour gérer le parallélisme et la concurrence, l'optimisation de la génération de binaire en assurant que les dépendances ne sont compilées qu'une fois.
Ce qui est intéressant dans Go est le support industriel de Google, mais la présence de cette grande entreprise pose des questions, Google ayant ses propres objectifs.

Thierry Goubier se demande s'il est sensé de prendre ce langage. Pour finir, il exprime son inquiétude sur le manque d'investissements européens dans le domaine et sur la dépendance aux grandes entreprises américaines.

---

2. Goland dans un produit de cybersécurité des réseaux industriels

Rémy Mathieu (Sentryo)

Rémy Mathieu est ingénieur R&D chez Sentryo, entreprise qui réalise des solutions contre les cyber-risques pour l'industrie. Elle veut montrer aux ingénieurs ce qui se passe dans leurs réseaux et les protéger.

Sentryo a adopté le langage Golang (Go), car ses besoins concernent le système, le métier et le web. Bien sûr, Python, C, C++ ou Javascript peuvent le faire, mais Golang était intéressant, car la librairie Gopacket était écrite en Go. C'est une librairie open-source créée par Google dont Sentryo avait besoin. Comme la librairie était en Go, l'entreprise a choisi de tout écrire en Go. Aujourd'hui 80 à 95% du code écrit à Sentryo est en Go. Pour un développeur, la simplicité de définition et de syntaxe de Go est un atout fondamental. Rémy Mathieu note aussi l'avantage de la gestion de la mémoire, celui de la mise en production, du déploiement et la concurrence qui est facilitée au cœur même du langage. Les packages standards sont utilisés en production dans de nombreuses entreprises. Ce sont eux qui changent les versions de Go qui n'a pratiquement pas bougé depuis Go 1.0. Un autre avantage est le formatage du code ou la génération de la documentation.
Rémy Mathieu note qu'il est encore difficile de recruter un développer Go, car le langage est peu présent dans les formations, même s'il est facile de former quelqu'un rapidement comme Sentryo l'a fait.

3. Go en physique des particules et en cosmologie : traitement de données, alertes et programmation parallèle

Sébastien Binet (CNRS)

Sébastien Binet est ingénieur de recherche au CNRS à l'IN2P3 (Institut nationale de physique nucléaire et de physique des particules) et au laboratoire de physique corpusculaire de Clermont-Ferrand.

Il commence par décrire son domaine et ce qu'il fait. La physique des hautes énergies s'occupe des lois fondamentales de l'Univers. Un des laboratoires connu dans le domaine est le Cern près de Genève où se trouve le LHC, le collisionneur le plus puissant du monde. Il est constitué d'un anneau formé d'aimants dipolaires qui accélère les protons. Quatre détecteurs sont installés à des endroits stratégiques de l'anneau. Atlas, par exemple, essaie de retrouver les traces de particules issues des chocs proton-proton. Les collisions se produisent toutes les 25 nanosecondes. Chacune génère 1 mégabits de données, ce qui fait 10 pétabits/an. Des outils ont été développés en amont pour donner une idée de ce que chaque détecteur peut voir. Il faut générer la production de chaque événement physique, simuler les interactions, reconstruire les particules qui ont donné les traces observées et enfin analyser les données pour tester les hypothèses. Les données arrivent directement dans la reconstruction qui s'aide de la simulation pour produire l'analyse.

Sébastien Binet revient ensuite sur le codage. Originellement, les codes étaient écrit en fortran-77, puis en C** au milieu des années 1990. Au fur et à mesure, tous ont migré vers le C++. Mais c'était lent à programmer. Python a pris le relai dans les années 2000,

tout en utilisant les bibliothèques en C++. Il faut savoir que chaque expérience du LHC regroupe 3000 physiciens dont 300 développeurs de code, de toutes les parties du monde. Chaque code représente environ 5 millions de lignes en C++ et 1 million de lignes en Python. C'est un gros volume à gérer. Le niveau des codeurs est très varié, des experts au jeune thésard. La pérennité du code est une réelle question.
Lancer une application demande une grosse minute et prend de la place (2 à 4 gigabits). Quand on lit toutes les *features* du langage Go, on voit que cela marche bien pour la physique des particules. C'est rapide et internationalement connu. C'est donc ce qu'il faut au Cern. Mais la migration de 5 millions de lignes de code quand l'expérience fonctionne n'est pas aisé. On a fait une simulation (expérience pre-Go 1.5) sur un plus petit programme. Appeler un programme C++ depuis go est facile. Cela prend du temps mais c'est faisable. Il faut recompiler toute l'application à la volée. Depuis aout 2015, avec Go 1.5, ce n'est plus la peine. Il n'y a pas de surcharge de l'opérateur, pas de *templates*. À l'usage, ce n'est pas une limitation.

Go est-il applicable au Cern ? Le compilateur GC compile rapidement mais ne peut pas tout faire. Le code sera moins performant que le C++, mais ce n'est pas un problème. Un programme sur une collision électron-positron qui prend 30 secondes en C++, a pris 26 sec en Go 1.5 et 23 secondes en Go 1.7. C'est donc plus rapide. Autre avantage, les applications fonctionnement en même temps. Sébastien Binet a fait des essais sur Linux, MacOS X et autres pour tester la différence entre Delphes en C++ et fads en Go 1.7 (beta2). Les performances sont meilleures dès qu'on dépasse 2 *threads*.
En cosmologie, Sébastien Binet a développé en Go une simulation de fusion du cœur d'une supernova ainsi qu'une application de contrôle-commande de télescope. Quelqu'un dans le labo a aussi écrit un programme d'acquisition de données dans le monde médical.

Comme Go est un langage neuf, il n'y a pas encore de support complet mais la communauté Go y travaille. Sabastien Binet note, pour finir, que la communauté Go en sciences ne se limite pas à la physique. Il y a la communauté biogo en biologie et gochem en chimie.

4. Go, LoRa, Go, ou comment Go accompagne la révolution IoT chez Bouygues Telecom

Jean-François Bustarret (Objenious)

Objenious est une filiale de Bouygues télécom, créée en novembre 2015, qui gère la technologie Lora, qui met en relation les objets et les applications métier par ondes radio. Les objets connectés peuvent être des pelleteurs, des capteurs, des détecteurs de fumées, des parkings, etc. Lora est un système ouvert, performant en sous-sol (mieux que le GSM qui doit être à l'air libre). C'est la révolution de l'IoT.

Début 2016, la société devait gérer 100 millions de capteurs, 150 000 messages par seconde, soit 2 pétabytes de stockage. Elle s'est posé la question de la plate-forme de gestion à mettre en place. Elle doit être éprouvée, mais elle doit réagir vite, être adaptée au besoin, ne pas être chère, mais sûre. L'idée a été de développer le système en interne. C'est rentable avec 4 ETP dès le premier capteur. Pour 1 million de capteurs, le point d'équilibre est de 67 développeurs. Les alternatives de développement sont le C++, mais ce langage n'est pas assez productif, il connaît des problèmes de gestion de mémoire, et de concurrence complexe. C'est pareil pour Java. Python n'est pas assez performant, JS/node n'est pas assez déterministe et difficile à maintenir. Rust, D, Scala ou Erlang sont trop exotiques, sans communauté. Ainsi Objenious a choisi Go, facile à apprendre, à coder, à maintenir, à exploiter. Go est plus efficace que Java surtout au niveau mémoire. Pour 2 millions de capteurs, en Java ou Python cela demande 20 serveurs, alors qu'il n'en faut que 8 en Go, sans les services de support. Autre avantage, Go communique bien par le réseau. La plate-forme est un regroupement de services avec un pipeline de données et des services concurrents.

Jean-François Bustarret reprend l'idée du peu de développeurs sur la marché, mais remarque que Go est facile à apprendre, productif donc demande peu de développeurs. La complexité est de faire des algorithmes faciles à maintenir.

5. Eléments de serveurs web

Valentin Deleplace (FREMN Corp)

Valentin Deleplace est développeur chez Cubiti. Durant toute sa communication il présente différents exemples concrets d'écriture de codes en Go. Il montre des éléments concrets de codes en langage Go.

Go est un langage compilé. Un code en Linux peut être écrit en Windows et en Darwin en 3 lignes de codes comme Valentin le montre avec la fonction simple chronometer. Il donne ensuite des exemples de goroutines (plusieurs fonctions en même temps ou Runconcurrent) avec gestion des compteurs. Il montre qu'une exécution de 3 routines à la suite se fait en 6 secondes alors qu'il faut 3 secondes avec Runconcurrent, car c'est lui qui lance les fonctions demandées.
Dans la programmation concurrente et le parallélisme, c'est plus facile en go. Valentin montre un exemple avec un simple serveur qui implémente un compteur interne. Il montre comment fonctionne Pprof, un CPU et *memory profiler*. Et termine en donnant quelques liens utiles comme le blog « *introducing the Go Race Detector* ».

L'après-midi commence dans l'amphi Becquerel par une intervention de Bruno Murati, de la société Beekast, qui s'invite en plus dans le séminaire pour présenter sa société. Beekast lutte contre le désintéressement lors des réunions, séminaires ou événements en entreprise. Il faut redonner la parole à l'audience et du pouvoir au présentateur. Avec leur smartphone ou leur ordinateur, les participants peuvent donner leur avis à travers app.beekast.com. On peut même faire un sondage en direct. Tests à l'appui.

6. Analyse de données de microcontrôleurs

Henri Lepic (Digikong)

Henri Lepic a un master de graphiste et est arrivé sur Go par hasard.

Il présente un projet de tapis sensible qui détecte les chutes de personnes, tapis qu'on trouve en ehpad (établissement d'hébergement pour personnes âgées) par exemple. Henri Lepic donne quelques chiffres concernant les chutes : 400 000/an, 12 000 décès, 50 000 cols du fémur cassés…. Le marché visé est 1,3 millions de logements, 21,4 millions de m^2 soit 21 millions de capteurs intégrés. Henri Lepic travaille dessus depuis 1,5 an. Go lui permet d'avancer. Il y a des contraintes pour le tapis : une basse consommation (12 volts), des données sûres, une capacité de 1To/jour récupérée par une box. L'infrastructure comprend 30 microcontroleurs par tapis dont les données sont envoyés à un analyseur. Dans Go, les constantes sont des nombres « fongibles » associés à des types différents (float, int). Il faut récupérer les données, les encrypter et les envoyer. Ces fonctions sont dans la librairie standard de Go. Il manque des choses en *machine learning*, ce qui a poussé la jeune société à écrire les codes adéquats en partant des constantes en float64. Henri Lepic note que dans le secteur de la santé, les normes à obtenir sont vitales. L'utilisation des tests unitaires est très facile dans Go et permet de comprendre comment fonctionne l'algorithme. Autre point intéressant, les benchmarks. Certaines parties peuvent être écrites en C pour optimiser, mais ce n'est pas testable.

Dernière chose critique, la documentation. Avec Go on peut la récupérer facilement. Avec sa formation de graphiste, Henri Lepic remarque que Go est bien designé.

7. Les apports de l'écosystème Go pour les challenges de l'ingénierie d'architectures clou microservices

Frédéric Ménez (Appsule Inc)

Frédéric Ménez vient des jeux vidéos qu'il a traité en C++. Il a senti que le *cloud gaming* allait bouger. Il a regardé les langages et s'est rendu compte que Go était intéressant à creuser. Il est aujourd'hui consultant indépendant, formateur en Go.

L'écosystème Go apporte beaucoup de choses à l'ingénierie d'architectures microservices. Un logiciel backend monolithique comme Magento ou Django a des sous-systèmes exécutables. C'est plus facile à penser, à programmer, à construire, corriger et déployer. Mais le couplage fort entre les composants induit des bugs, des décalages d'implémentation, des erreurs de compilation. Cela peut créer de la frustration dans les équipes de programmeurs. S'il y a un crash, le public est mécontent. Pour pallier ces problèmes, il faut décentraliser, diviser pour régner vers des architectures encore plus distribuées. Après les promesses du SOA, les microservices fonctionnent. Ces architectures microservices comprennent plusieurs processus qui communiquent ensemble. C'est une architecture qui s'adapte au business. Le nombre de microservices peut augmenter ou diminuer. Un microservice se limite à un périmètre métier retreint pour une tache précise. Il est composable avec d'autres. Comme chacun est indépendant, ils peuvent être programmés en n'importe quel langage. Les logiciels microservices sont livrés plus rapidement. Ils facilitent l'échelonnabilité, rendent autonome les équipes.

L'infrastructure cloud est capable de refaire l'architecture en cas de défaillance d'un microservice. Mais tout n'est pas rose. Un microservice est lié à son environnement. Et crée de nouveaux besoins, de nouvelles contraintes. Le process de construction du code doit être plus élaboré. Et comme on augmente le risque de failles, il faut intégrer les échecs dans la conception du code général. On peut trouver du Go à tous les étages de l'architecture microservices. Il s'installe en 3 minutes, le code est récupéré très vite contrairement à Java ou C++, et l'exécution est rapide. Il peut y avoir des délais entre les logiciels de microservices, mais l'asynchronie du Go est native. L'outillage réseau existe (http, rest, etc.), comme les outils d'analyse. Des boites à outils flexibles existent comme go-kit, gizmo, go-nano, go-micro ou Armor. Go-kit facilite le transport de lignes de Java en Go. Frédéric Ménez donne un exemple de la création d'un client microservice GRPC.

Il décrit ensuite des exemples de différents outils (drone.io, consul), les solutions d'orchestration conçus avec Go comme Kebernetes (800 contributeurs) et Docker Swarm, ainsi que les outils de traçabilité et de métrique (prometheus, Heka, snap, grafana).

8. Le langage Go dans le calcul HPC Big Data

Louiza Hanis (Inra)

Louiza Hanis suit un stage de M2 à l'Inra sur la métagénomique. C'est le big data de l'ADN bactérien.

Pour séquencer l'ADN des fèces, il faut extraire des clusters de manière exhaustive et déterministe. On obtient des matrices qu'il faut analyser. Il faut faire un code HPC spécifique unifié, des tests unitaires, d'intégration, de vérification. C'est fait en langage R, mais en Go ce serait mieux. Elle a utilisé Cayley, un moteur de requêtes de Google, écrit en Go et a comparé le temps mis en R, en Go et en goroutines. Go et goroutines sont beaucoup plus efficaces.

Après avoir codé en C et C++, Louiza Hanis remarque que les avantages de développer en Go sont multiples : un environnement complet, l'intégration de plusieurs outils et des packages faciles à importer comme Goget.