Go-ning qurilishi

(Bu 2012 yil 25-oktabrda Arizona shtatining Tusson shahrida bo'lib o'tgan SPLASH 2012 konferentsiyasida Rob Payk tomonidan berilgan asosiy nutqning tahrirlangan versiyasi.)

Abstract

Biz Googleda dasturiy ta'minot infratuzilmasi rivojlanishida ko'rgan ba'zi muammolarimizga javob sifatida 2007 yil oxirida Go dasturlash tilini yaratdik. Bugungi kunning dasturlash landshafti ko'p foydalanilayotgan C++, Java va Python tillariga bog'liq emas. Chunki ko'p yadroli protsessorlar, tarmoq tizimlari, ulkan hisoblash klasterlari va veb-dasturlash modeli tomonidan kiritilgan muammolar boshdan-oyoq ko'rib chiqilmagan, balki atroflicha ishlangan. Bundan tashqari, o'sish ko'lami o'zgargan: bugungi server dasturlari o'n millionlab satrlarni o'z ichiga oladi, yuzlab yoki hatto minglab dasturchilar tomonidan ishlanadi va har kuni so'zma-so'z yangilanadi. Eng yomoni, katta kompilyatsiya klasterlarida ham kodni kompilyatsia vaqtlari juda ko'p daqiqalarga, hatto soatlarga cho'zilgan.

Go tili ushbu muhitda ishlashni yanada samarali qilish uchun ishlab chiqilgan. Go-ning dizayni Concurrency va Garbage-Collection kabi taniqli jihatlaridan tashqari, Dependency Management, tizimlar o'sib borishi bilan birgalikda dasturiy ta'minot arxitekturasiga moslashuvchanlikni va Componentlar o'rtasidagi mustahkamlikni o'z ichiga oladi.

Ushbu maqolada quyida qayd etilgan masalalar kompilyatsiya qilingan dasturlash tilini yaratish paytida, qanday qilib yengil va samarali hal qilinganligi tushuntiriladi.

Kirish

Go - bu Google-da ishlab chiqilgan, kompilyatsiya bo'ladigan, garbage-collected, static typed dasturlash tili. Bu ochiq manbaali loyiha bo'lib, Google ochiq reposiztlarni import qiladi, aksincha emas.

Go effektiv, samarali va yuqori masshtabga ega til. Ba'zi dasturchilarga ishlash qiziqarli bo'ladi; boshqalar qandaydir tasavvurga ega emas, hatto zerikarli deb bilishadi. Ushbu maqolada nima uchun ular qarama-qarshi pozitsiyalar emasligini tushuntiramiz.

Go Google-da

Go Google-da dasturiy ta'minotni ishlab chiqishda yuzaga kelgan muammolarni hal qilish uchun ishlab chiqilgan bo'lib, bu ilmiy tadqiqot tili bo'lmagan, ammo baribir katta dasturiy ta'minot loyihalarini ishlab chiqish uchun ajoyib vosita bo'lgan.

Google katta muammolarni hal qilishda yordam beradi va Google katta muammolarga duch keladi. Hardware (uskunalar qisimi) va Software (dasturiy ta'minot) katta. Serverlari asosan C++ da va boshqa qismlari Java va Python da yozilgan millionlab dasturiy ta'minot liniyalari mavjud. Minglab muhandislar barcha dasturlarni o'z ichiga olgan bitta daraxtning "boshida" kod ustida ishlaydi, shuning uchun kundan-kunga daraxtning barcha darajalarida sezilarli o'zgarishlar yuz beradi. Xususiy tarqatilgan tizim (custom-designed distributed build system) miqyosda rivojlanishni amalga oshirishi mumkin, ammo bu hali ham katta.

Va albatta ushbu dasturiy ta'minotning barchasi millionlab mashinalarda, tarmoqli hisoblash klasterlari yordamida amallarni mustaqil bajaradi.

Qisqa qilib aytganda Google da rivojlanish katta. Bu sekin va noqulay bo'lishi mumkin. Ammo bu samarali.

Go loyihasining maqsadi Google-da dasturiy ta'minotni ishlab chiqarishdagi sustlik va beparvolikni yo'q qilish va shu bilan jarayonni yanada samarali va ko'lamini oshirishga qaratilgan edi.

Til katta dasturiy ta'minot tizimlariga xizmat ko'rsatadigan ularni yaratadigan va rijovlantiradigan odamlar tomonidan ishlab chiqilgan.

Shuning uchun Goning maqsadi dasturlash tili dizaynini tadqiqot qilish emas; Uning dizaynerlari va ularning hamkasblari uchun ish muhitini yaxshilashdir. Tadqiqotlar qilishdan ko'ra ko'proq dasturiy ta'minot muhandisligiga qaratilgan. Yoki dasturiy ta'minot yaratishda yordam beruvchi xizmat hisoblanadi.

Ammo til dasturiy ta'minotga qanday yordam berishi mumkin? Ushbu maqolaning qolgan qismi bu savolga javob beradi.

Og'riqli nuqtalar

Go ishga tushirilganda, ba'zilar zamonaviy til uchun odatiy deb hisoblangan ba'zi xususiyatlar yoki metodikalar etishmayotganligini da'vo qilishdi. Ushbu imkoniyatlar bo'lmagan taqdirda Go qanday qilib qadrli bo'lishi mumkin? Bunga javobimiz shundaki, Go keng ko'lamli dasturiy ta'minotni ishlab chiqishdagi qiyinchiliklarni hal qilish xususiyatiga ega. Ushbu masalalarga quyidagilar kiradi:

- Slow builds
- Dependency Management
- Har bir dasturchi tildagi har xil subset dan foydalanishi
- Dasturni yomon tushunish (kodni o'qish qiyin, hujjatsiz va hokazo)
- Harakatlarning takrorlanishi
- Cost of updates (Yangilanishlar narxi)

- Version Skew (Versiyadi cheklovlar)
- Avtomatik vositalarni yozish qiyinligi
- Cross-language builds (Tillararo qurilish)

Tilning individual xususiyatlari bu muammolarni hal qilmaydi. Katta dasturiy muhandislikda talab etilganidek, Go dizaynida biz ushbu muammolarning echimlariga e'tibor qaratishga harakat qildik.

Oddiy, o'ziga xos misol sifatida dastur tuzilmasining ko'rinishini ko'rib chiqamiz. Ba'zi kuzatuvchilar Go-ning C-ga o'xshash blok tuzilishiga tanqidlar bilan qarshi chiqdilar va Python yoki Haskell uslubida bo'shliqlardan foydalanishni afzal ko'rishdi. Shu bilan birga SWIG dan foydalanishda boshqa tilga o'rnatilgan (embedded) Python kodning indentatsiyasi o'zgartirganligi (buziliganligi) sababli, tillararo build qilish natijasida hosil bo'lgan muammolar va sinov jarayonidagi xatolarni kuzatishda katta tajribaga ega bo'ldik. Shuning uchun bizning pozitsiyamiz shundan iboratki, identatsiyalar kichik dasturlar uchun yaroqli bo'lsa ham, uni rivojlantirish qiyin va kod bazasi qanchalik katta va xilma-xil bo'lsa, shuncha ko'p muammolarga olib kelishi mumkin. Xavfsizlik va ishonchlilik uchun qulaylikdan voz kechgan ma'qul, shuning uchun Go-da mahkamlangan bloklar mavjud.

C va C++ tillarida Dependency Management

Dependency Management bilan ishlashda boshqa muammolar paydo bo'ladi. Biz ularning C va C++ da qanday ishlashini ko'rib chiqish bilan muhokamani boshlaymiz. Dastlab 1989 yilda standartlashtirilgan ANSI C standart sarlavha fayllarida #ifndef "guards" g'oyasini ilgari surdi. Hozir hamma joyda tarqalgan g'oya shundan iboratki, har bir sarlavha faylini shartli kompilyatsiya bandi bilan qo'yish kerak,

shunda bir fayl xatosiz bir necha marta kiritilishini (ulanishini) oldini oladi. Yoki misol uchun Unix sarlavha fayli <sys/stat.h> quyidagicha sxematik ko'rinishga ega:

```
/* Large copyright and licensing notice */
#ifndef _SYS_STAT_H_
#define _SYS_STAT_H_
/* Types and other definitions */
#endif
```

Maqsad shundaki, C preprocessor faylni o'qiydi, lekin faylning ikkinchi va keyingi o'qishidagi tarkibni inobatga olmaydi. Faylni birinchi o'qishda aniqlangan *SYS_STAT_H* belgisi, keyingi chaqiruvlarni "himoya qiladi". Ushbu dizayn ba'zi bir yaxshi xususiyatlarga ega, eng muhimi, har bir sarlavha fayli barcha bog'liqliklarni xavfsiz ravishda # o'z ichiga olishi mumkin, hatto boshqa sarlavha fayllari ham ularni o'z ichiga oladi. Agar ushbu qoidaga rioya qilinsa, masalan, #include bandlarini alifbo tartibida tartiblaydigan tartibli kodga ruxsat beriladi. Ammo bu juda yomon.

1984 yilda Unix ps buyrug'ining manbai bo'lgan ps.c kompilyatsiyasi, barcha qayta ishlash jarayoni tugaguniga qadar #include <sys/stat.h> ni 37 marta kuzatgan. Shunday qilib, kodning tarkibini 36 marta tashlab o'tsa ham, C dasturlarining aksariyati faylni ochadi, o'qiydi va 37 marta skanerlaydi. Darhaqiqat, katta aql-idrokka ega bo'lmagan holda, bu xatti-harakatni C protsessorining murakkab semantikasi talab qiladi.

Dasturiy ta'minotga ta'siri, C dasturlarida #include bandlarini asta-sekin yig'ishdir. Ularni qo'shish dasturni buzilmaydi va ular qachon kerak bo'lmasligini bilish juda qiyin. #Include-ni o'chirish va dasturni qayta kompilyatsiya qilish, hatto uni sinab ko'rish uchun ham etarli emas, chunki boshqa #included-ning o'zida o'zi uchun kerakli yana boshqa #include kiritadigan bo'lishi mumkin.

Texnik jihatdan bunday bo'lishi shart emas. #Ifndef dan foydalanish bilan bog'liq uzoq muddatli muammolarni anglab etgan holda, Plan 9 kutubxona dizaynerlari boshqacha, ANSI standartiga mos bo'lmagan usulni qo'lladilar. Plan 9 ga ko'ra, sarlavha fayllarida yana #include bandlarini kiritish taqiqlangan; Barcha #include lar C faylining yuqori qismida bo'lishi kerak edi. Buning uchun, albatta, intizom kerak edi - dasturchi kerakli bog'liqliklarni aniq bir marta, to'g'ri tartibda ro'yxatlashi kerak edi, ammo hujjatlar yordam berdi va amalda u juda yaxshi ishladi. Natijada, C manba faylida qancha bog'liqlik bo'lishidan qat'i nazar, har bir #include fayli ushbu faylni kompilyatsiya qilishda bir marta aniq o'qiladi. Va, albatta, #include-ni olib tashlash kerakligini bilish ham oson edi.

Plan 9 yondashuvining eng muhim natijasi tezroq kompilyatsiya bo'ldi: kompilyatsiya talab qilinadigan kiritish-chiqarish miqdori #ifndef dan samaraliroq bo'ldi. Plan 9 dan tashqari, C va C ++ uchun "guards" yondashuv qabul qilingan.

An'anaga ko'ra C ++ dasturlari odatda bitta class uchun bitta sarlavha fayli yoki <stdio.h> guruhlarga bo'lingan class larning kichik to'plami bilan tuziladi. Shuning uchun Dependency Management ancha murakkab bo'lib, kutubxonaga bog'liqlikni emas, balki to'liq ierarxiyani aks ettiradi. Bundan tashqari, C ++ sarlavhali fayllar odatda C kodi fayliga xos oddiy konstantalar va funktsiya imzolarini emas, balki haqiqiy kod - turi, usuli va shablon deklaratsiyalarini o'z ichiga oladi.

C++ kompilyatorga ko'p ma'lumot berish bilan, har bir ma'lumotni qayta ishlashga majbur qilib kodning compile bo'lishini qiyinlashtiradi.

Katta C ++ dasturini tuzishda kompilyator <string> sarlavha faylini qayta ishlash orqali string ni qanday ifodalashni minglab marta skaner qilishi mumkin.

C ++ ikkilik tizimining tuzilishida Google yuzlab individual sarlavha fayllarini o'n minglab marta ochishi va o'qishi mumkin. 2007 yilda Google-da muhandislar Google-ning asosiy ikkilik dasturini tuzdilar. Unda ikki mingga yaqin fayl bor edi, agar ular birlashtirilsa, jami 4,2 megabaytni tashkil etadi. #Include lar kengaytirilgunga qadar kompilyatorning kirish qismiga 8 gigabaytdan ko'proq etkazib berildi, bu har bir C ++ manba kodi uchun 2000 baytni tashkil etdi.

Ma'lumot o'rnida, 2003 yilda Google-ning build tizimi bitta Makefile yaxshiroq boshqariladigan va aniq bog'liqliklarga ega bo'lgan katalog boshiga ko'chirildi. Bu oddiy ikkilik fayllar hajmini taxminan 40% ga qisqartiradi, shunchaki aniqroq bog'liqliklarni qayd etilganligidan. Shunga qaramay, C++ da (yoki bu o'rinda C) ushbu bog'liqliklarni avtomatik ravishda tekshirish maqsadga muvofiq emas va bugungi kunda biz hali ham katta Google C ++ ikkiliklarining qaramlik (Dependency Management) talablarini aniq tushunmayapmiz.

Ushbu nazoratsiz bog'liqliklar va katta miqyosda ishlash natijasi shundaki, Google serverining ikkilik fayllarini bitta kompyuterda yaratish maqsadga muvofiq emas, shuning uchun katta tarqatilgan kompilyatsiya tizimi yaratildi. Ko'pgina mashinalarni, juda ko'p keshlashni va juda murakkablikni o'z ichiga olgan ushbu tizim yordamida (tuzish tizimi o'z-o'zidan katta dastur), Google-da qurish juda qiyin bo'lsa ham, hozirgi paytgacha amalda.

Hatto tarqatilgan qurilish tizimida ham katta Google qurilishi bir necha daqiqa davom etishi mumkin. 2007 yildagi ikkilik taqsimlangan qurilish tizimidan foydalangan holda 45 daqiqa davom etdi; o'sha dasturning bugungi versiyasi 27 daqiqa davom etadi, ammo albatta dastur va unga bog'liqliklar vaqt oralig'ida o'sdi. Qurilish tizimini kengaytirish uchun talab qilinadigan muhandislik kuchi u yaratayotgan dasturiy ta'minotning o'sishida katta yordam berolmadi.

Go-ga o'tish

Qurilishlar sekin bo'lsa, o'ylash uchun vaqt bor. "Go" ning kelib chiqishi haqidagi afsonada aytilishicha, Go 45 daqiqali qurilishlardan birida o'ylab topilgan. Bu veb-serverlar kabi yirik Google dasturlarini yozish uchun mos bo'lgan yangi tilni ishlab chiqishga va Google dasturchilarining hayot sifatini oshiradigan dasturiy ta'minot muhandisligi bilan ishlashga arziydi.

Garchi munozara shu paytgacha bog'liqliklarga bag'ishlangan bo'lsa-da, boshqa ko'plab muammolar e'tiborga muhtoj.

Ushbu kontekstda muvaffaqiyat qozonish uchun har qanday til uchun asosiy fikrlar:

- Bu katta miqdordagi bog'liqliklarga ega bo'lgan katta dasturlar uchun va ular ustida ishlaydigan dasturchilarning katta guruhlari bilan ishlashi kerak.
- Bu tanish bo'lishi kerak, taxminan C ga o'xshash. Google-da ishlaydigan dasturchilar o'zlarining martabalarini protsessual tillardan boshlashgan va xususan, C oilasidagi tillarni yaxshi bilishgan. Dasturchilarni yangi tilda tezda unumli ishlash zarurati, bu til juda radikal bo'lishi mumkin emasligini anglatadi.
- Bu zamonaviy bo'lishi kerak. C, C++ va Java ma'lum darajada eskirgan, ular ko'p yadroli mashinalar paydo bo'lguncha, tarmoqlar yaratish va veb-dasturlarni ishlab chiqishdan oldin yaratilgan. Zamonaviy dunyo yangi yondashuvlarni masalan Concurrency kabi xususiyatlarni talab etadi.

Muhammadxon Najimov (Najot Ta'lim) 2021

Davomi bor...