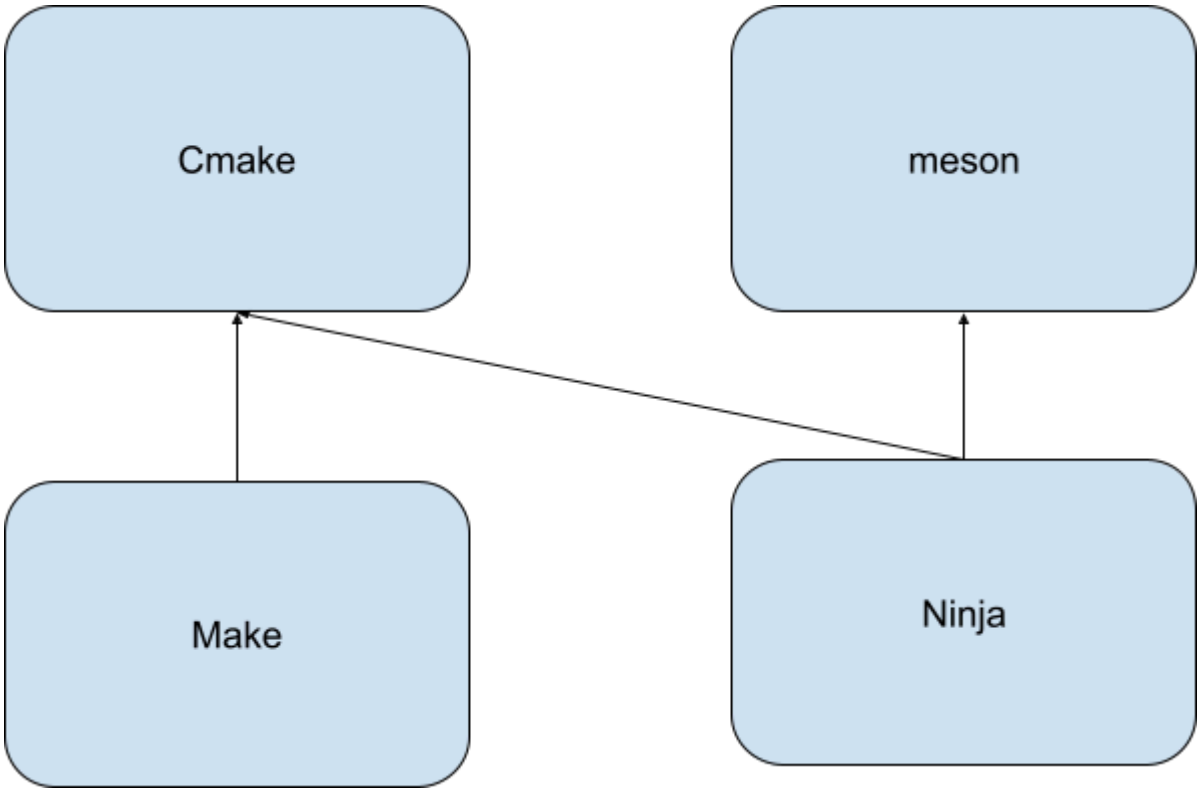
Bazel 정리

Build System 전반적인 내용 정리

CMake, Meson, Make, Ninja



turing-complete, non-turing complete
overview + Feature + examples simple

Make : shell script : turing complete

CMake - Meson - Cson
CMake : Make 의 의존성을 모두 관리하지 않아도, 알아서 기록되게. 추상화.
즉, 최종 결과물과 인풋만 주면 알아서 하게끔 함.
예제 https://www.tuwlab.com/ece/27234

Meson : non turing complete DSL
https://mesonbuild.com/GuiTutorial.html
쉽다.
ninja 이용

scons
https://github.com/SCons/scons-examples/blob/master/shared-lib-program/SConstruct
파이썬 기반. 느리다. 쉽다.

webUI

Bucks

bazel 에 영향을 많이 받음.
https://github.com/airbnb/BuckSample/blob/master/BuckLocal/BUCK

Pants
https://www.pantsbuild.org/2.18/docs/introduction/how-does-pants-work
거의 비슷하다.
remote execution 존재.

Bazel 정리

cache difference

make cache != bazel cache

Make cache : by Last-Modified Timestamps
(https://web.archive.org/web/20160813235106/http://www.conifersystems.com/whitepapers/gnu-make/)
Ninja cache : by Last-Modified Timestamps
(https://github.com/ninja-build/ninja/issues/1394)
Maven cache : digest based ,
but slow and completely incremental. why? task based! hard to find
important how to make key. input, Output, plugin
(https://maven.apache.org/extensions/maven-build-cache-extension/)
opt in feature

https://en.wikipedia.org/wiki/Bazel_(software)

bazel cache : default feature, hash based: content digest
gradle cache : default feature, hash based: content digest

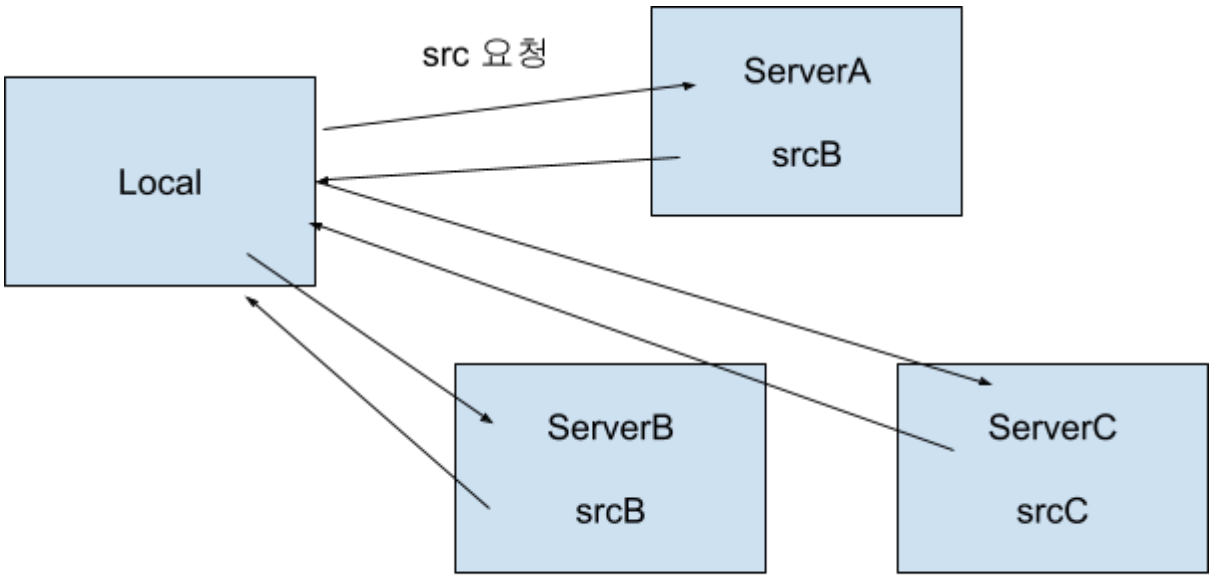
Q: Greenwich?

Bazel’s ways to increase performance build

distributed build.

with remote build(execution)
TODO
https://google-engtools.blogspot.com/2011/09/build-in-cloud-distributing-build-steps.html
4개 시리즈 전부 읽을 것. 배경을 다루었음.

bazel 블로그 .docs읽을 것.
1편 어떻게 소스를 가져올 것인가? 외부 dependency
과거



CI 도 마찬가지.

build from src 임. 기본적으로.
장점. compatibility problem 없음. 그런데 너무 오래걸림. too long
이를 해결하기 위해 dependency graph 를 계산해서 일부만 가져오게 바꿈. 효과 good
그런데, graph 계산 하는 시간, server 접근 시간이 그래도 조금 걸림.

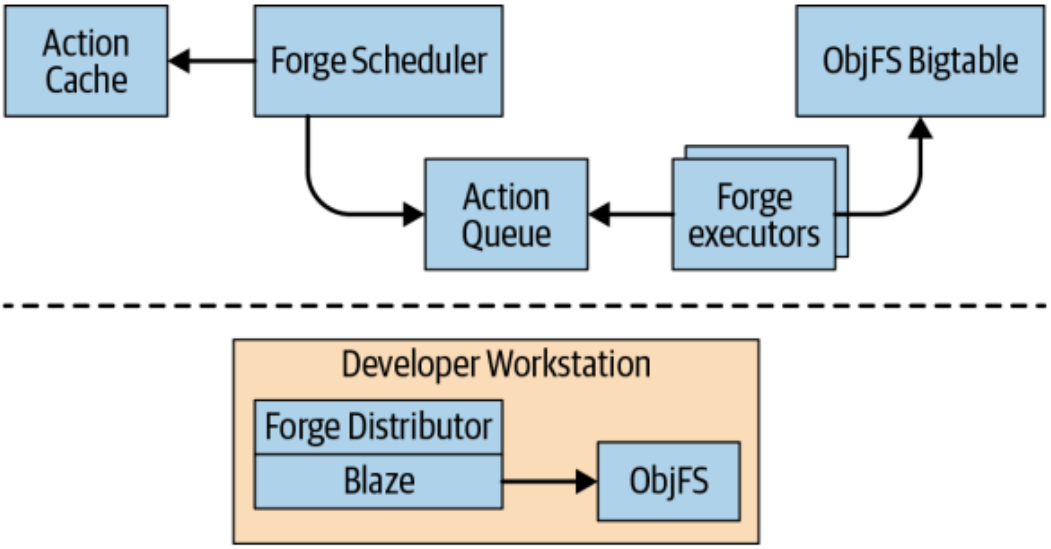


Figure 18-4. Google's distributed build system

[1]

위 책에 os 문제를 toolchain 으로 해결했다고 하는데, 이를 좀 보도록 하자.

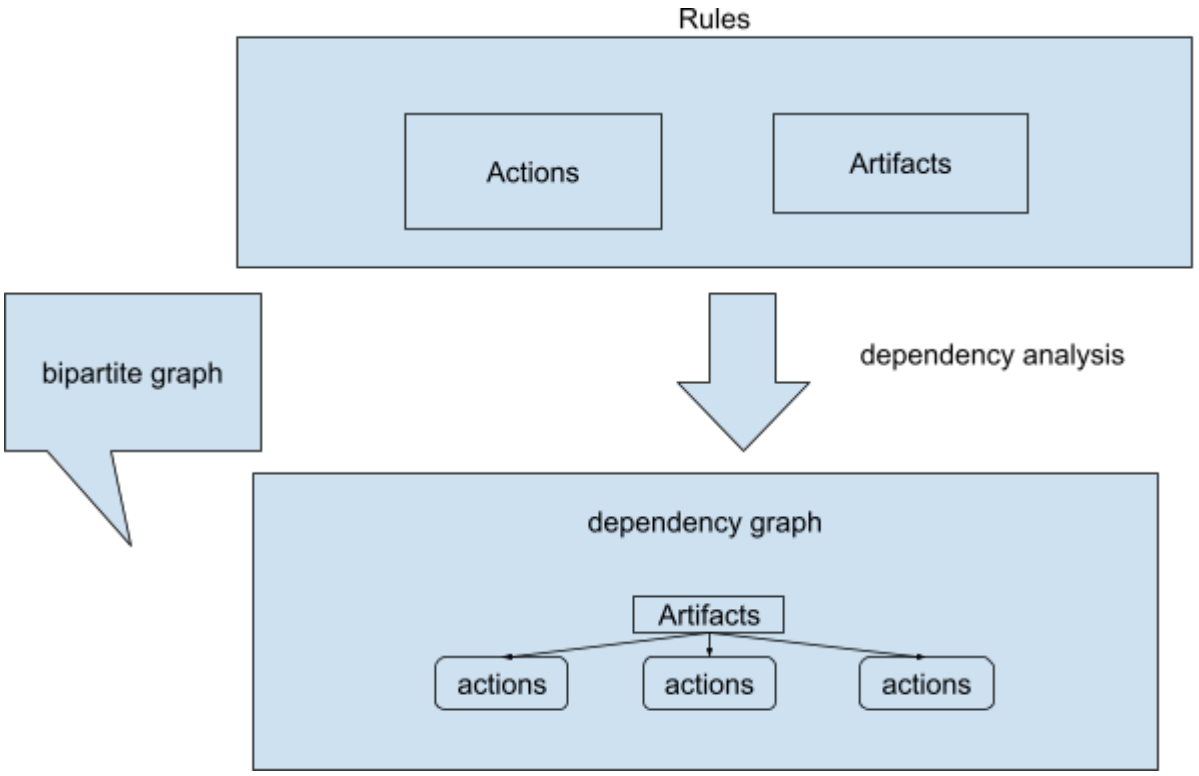
Forge : job 관련해서 일을 나누는 것 (bazel 이 함)
Forge Scheduler 는 이를 cloud 환경에 뿌리는 것.
Action cache 를 이용해 일을 여러번 하지 않음.
Forge Executors : cloud workers임
ObjFS 가 FUSE daemon 을 의미함.

저 bigTable 에 metadata, 와 content digest, content 저장.
CAS 로 활용하는 것임.

추가로, 1번 주제는 어떻게 소스를 가져올 것인가 인데,
이를 mirror All versions, 즉 한 군 데에 모두 모아서 복제 해서 관리하는 방식으로 바꾸었다고 함. [2]

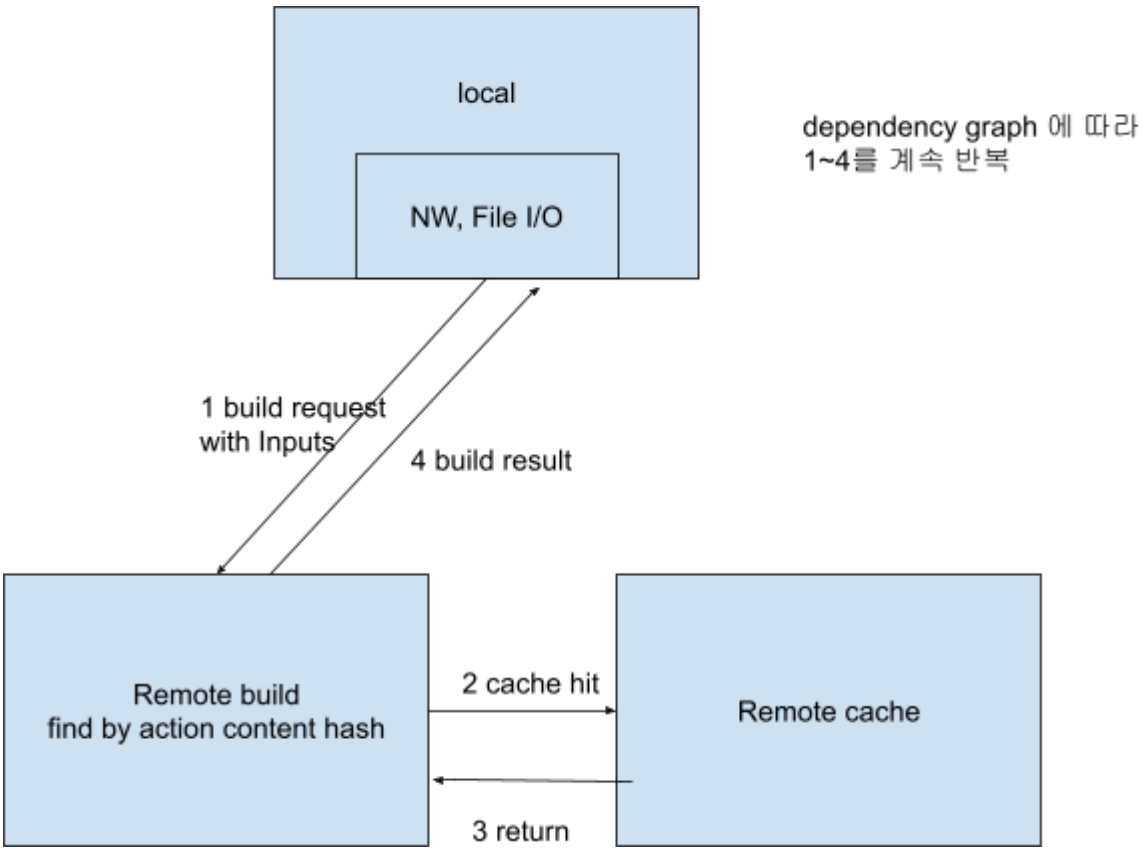
2편, 3편

[1] software engineering at google
[2] https://www.youtube.com/watch?v=W71BTkUbdqE



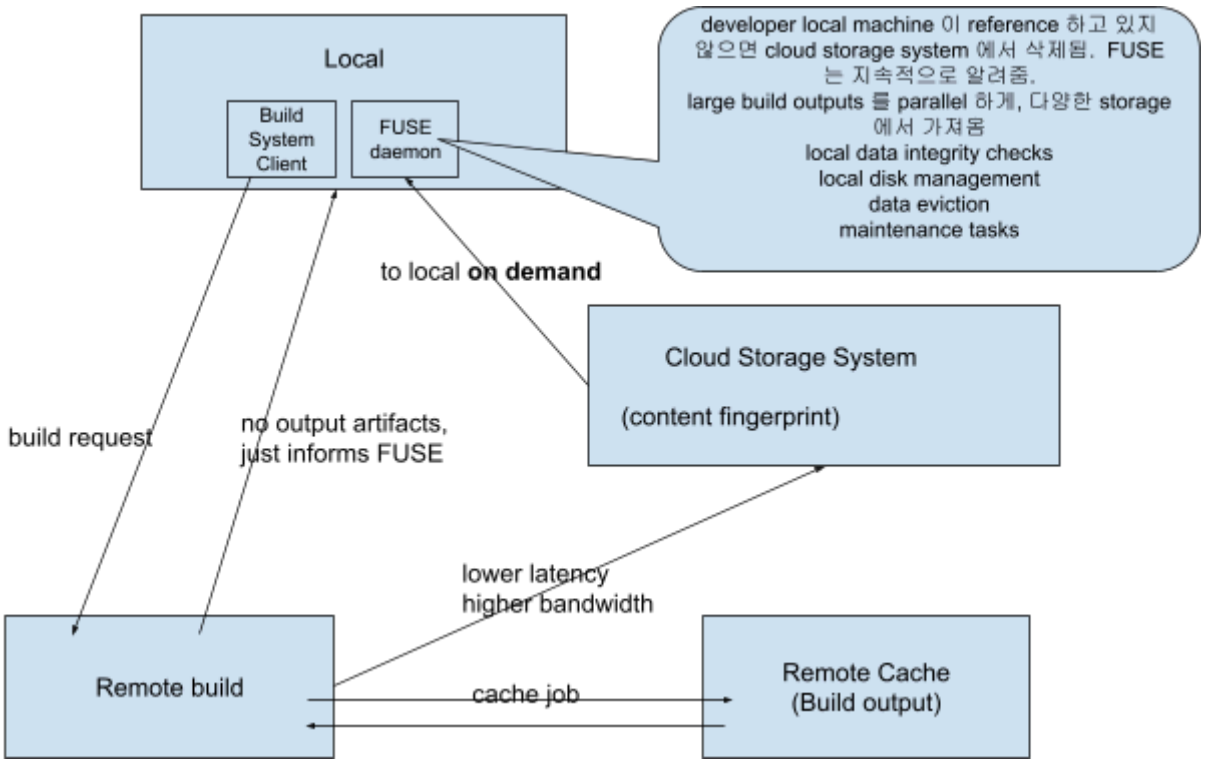
- content-based (not file, timestamp) - content digest key 로 쉽게 CAS 접근.
- dependency graph
- functional model
- Output = Actions (Inputs, Other parameters)
- key = f (Actions, Inputs(contents digests), Other parameters) 를 통해 output 을 가져올 수 있음
- hermetic
- build actions are portable
- white space 같은 간단한 수정의 경우, 첫 단계에서는 오래걸릴 수 있지만, 최종 output 이 한 번 결정되고 나면 artifacts 가 동일해지기 때문에 빠르게 처리할 수 있음.
  덧) maven 의 경우 그렇지 않음. 할 때마다 다시함.

이를 이용해 remote cache + remote execution 을 할 수 있음. 문제는, 매번 이 결과물들을 가져오고, input들을 보내고 하는 과정이 network 와 disk I/O를 거치며 매우 느려질 수 있다는 것임.

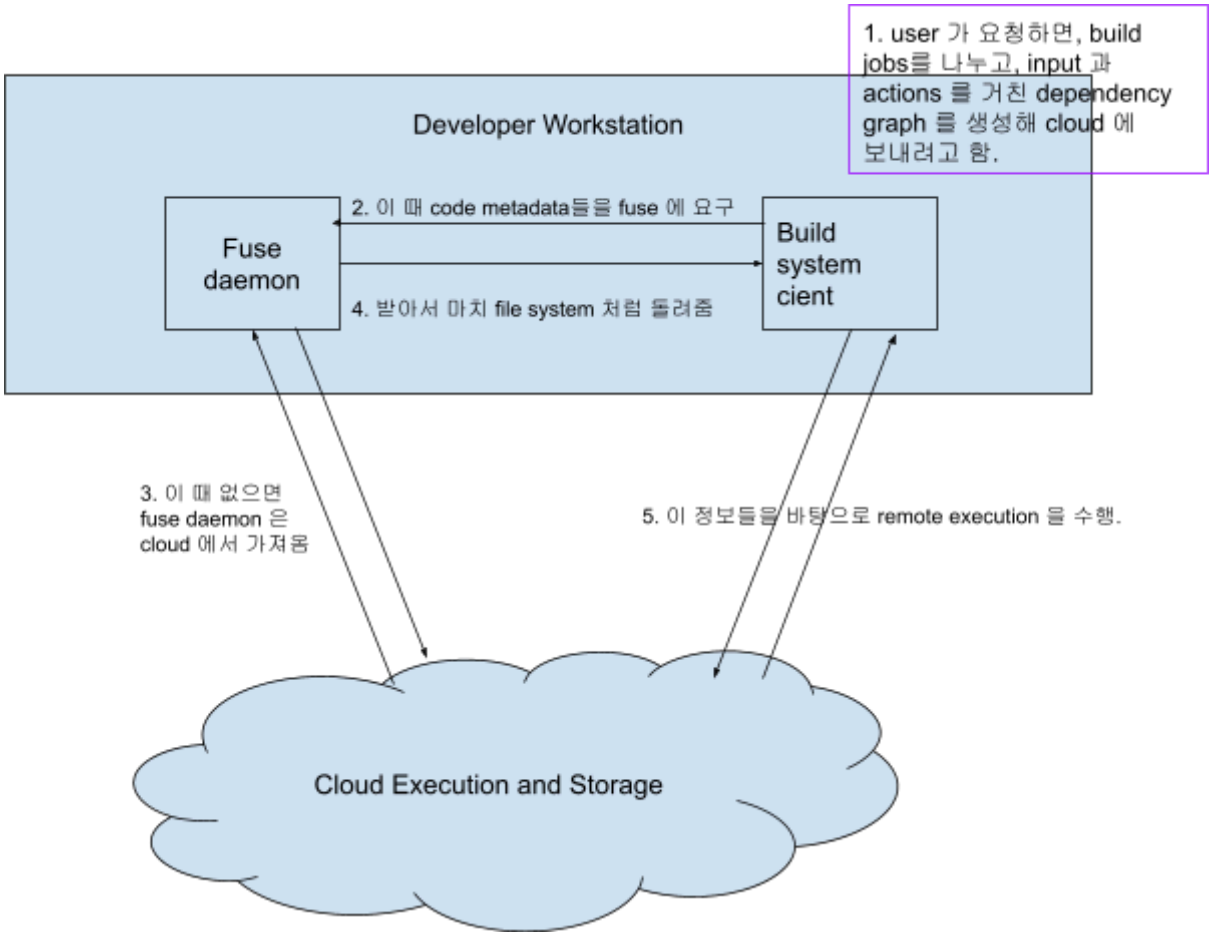


4편.
수 GB artifacts 들의 input, output 이동은 성능 저하 요소임.
게다가, developer 는 중간 outputs 을 모두 필요로 하지 않음. 테스트의 경우는 pass, fail 만 보고 싶어하는 경우임.
그러면 그럴 때마다 cloud 에 넣고 빼고를 반복해야 함.

그래서. Cloud Storage System 을 한 개 더 놓음.



local workspace 에서 build system client 와 fuse daemon 이 하는 역할을 조금 더 나타내 보자.



관련해서 single repository 넣는 내용, [3]
google CI tool 내용 유투브[4]

Google 의 Build 처리 과정

총 4편의 글, 그리고 강의가 있습니다. [5][6]
구글의 코드 개발은 엄청나게 빠르고, 용량이 거대함.
built from head 시스템임.

그런 시스템을 할 때마다 빌드와 테스트를 하는 일은 매우 힘든 일임
그래서 dependency analysis 를 도입함.
tests 간에 in memory graph 를 도입했음. 수 GB
dependency graph

이 dependency graph 를 작성하고, 관리하는 것도 쉽지 않음. 시간도 걸리는 일임.

그런데, 대부분의 개발이 일부 코드만 수정한다는 사실을 알게 되었고,

CAS Content Addressable Storage
This digest is the hash of the file content itself, created using a hashing function appropriate for Content Addressable Storage (CAS).
Our system watches for changes arriving in the version control system. As changes arrive, we hash the file contents to compute digests and insert the content into BigTable.
Bigtable is a fully managed wide-column and key-value NoSQL database service for large analytical and operational workloads as part of the Google Cloud portfolio.

TODO: FUSE, Bigtable

파일의 정보를 hash 함. digest 와 CAS 를 이용.
Action + input/output = rule
bipartite graph

TODO: software engineering at google 에서 문제 해결법을 다시 읽어볼것.

Software engineering at google. nifty way to handle problems

Tools as Dependencies.

platform dependency problem by toolchain

---

[3] https://www.youtube.com/watch?v=W71BTkUbdqE
[4] https://www.youtube.com/watch?v=KH2_sB1A6lA
[5] https://google-engtools.blogspot.com/2011/06/build-in-cloud-accessing-source-code.html
[6] https://www.youtube.com/watch?v=b52aXZ2yi08

Extending the build system

adding custom rules

Isolating the environment

sandboxing 기능을 이용.
TODO: https://blog.bazel.build/2017/08/25/introducing-sandboxfs.html
일부 기능에만 있다고 함. Linux Docker.

Making external dependencies deterministic

그럼에도, java 같은 경우.. 아니면 기타 library 는 직접 library 를 사용해야 함.
building 하는 것이 아니라.
굉장히 risky 한 행위임.
내부 dependency 가 바뀔 수도 있기 때문임. security risk.

workspace 레벨에서 cryptographic hash 를 사용하는 것.

Managing dependencies

internal dependencies : transitive 를 허용했다가, refactoring 과정에서, 수많은 간접참조들이 문제를 일으킨다는 사실을 발견했고, refactoring 이 너무 오래 걸리게 됨. (많은 라이브러리들을 살펴보며 간접참조 영역을 추가해야함)
Strict transitive dependency mode 를 추가함.
Rolling this change out across Google's entire codebase and refactoring every one of our millions of build targets to explicitly list their dependencies was a multiyear effort, but it was well worth it.

불필요한 dependency 도 제거되었고, dependency 걱정을 하지 않아도 됨(명시적임)
그러나 단점이 있음. 파일이 너무 길어짐.
그래도 1번 해놓으면 나중에 편하다. 는 생각으로 강제함.

external dependencies: version 문제가 있음.
manually vs automatically 로 구분됨.
당연히 manually를 선택.
One version rule : shading 기능 제공하지 않음. multi version 도 있음.
For this reason, Bazel does not automatically download transitive dependencies
Yet again, the choice here is one between convenience and scalability

Caching

remote caching

Build without the bytes?

bazel 7.0 feature

local repository?

Bazel grpc

Bazel basic concepts

Bazel BzlMod

transitive 안써도 되는지?
글 읽고 정리해보기
diamond collision 그래서 어떻게 해결했다는 건지? shaded?

rule 짜보기

Gradle 정리

Remote System 글 요약[7]

앞 내용들

gradle 은 cache 시스템 2017년에 제공 시작.

gradle, remote execution? distributed execution?
현재 제공하고 있지 않음. 그렇지만 다른 툴들이 존재함.
https://blog.gradle.org/remote-and-distributed-build-patterns

remote execution vs distributed execution 미묘한 개념 차이가 있는데, 정리해 보자.
First of all, Both are designed for Performance.
Remote build : 원격으로 build
remote execution : 원격으로 연결. vs code - remote IDE, IntelliJ IDEA, Fleet

---

[7] https://blog.gradle.org/remote-and-distributed-build-patterns

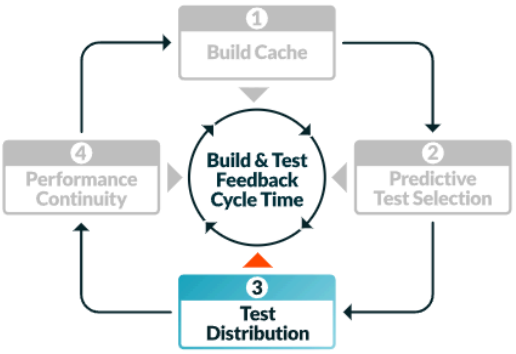
Cloud Development Environment : 빌드를 로컬에서 할 필요가 없음. 중앙 집중형. : vs code 로 뭐 있었던 것 같은디? Github codespaces, JetBrains Sapce, Gitpod
remote build 에서 remote execution, cloud development Environment
distributed build : 분산 build , 앞에서 다루었던 것들은 single remote machine. distributed build 는 multi machine 개념임.

Don't forget about the backing infrastructure needs of distributed builds.
A build could actually be slower if sufficient remote build agents are not available.

구글의 system 은 어떻게 되어 있는가?
구글 엔지니어는 이렇게 일한다에서 참고.

gradle 에서는 build, 특히 test build 에 대하여 다음 4단계로 단계를 나누어서 구분하고 있음.[8]

1. **Build Cache**. Avoid unnecessarily running components of builds and tests when inputs have not changed.
2. **Predictive Test Selection**. Run only tests that are likely to provide useful feedback using machine learning.
3. **Test Distribution**. Run the necessary and relevant remaining tests in parallel to minimize build time.
4. **Performance Continuity**. Sustain Test Distribution and other performance improvements over time with data analytic and performance profiling capabilities.



① **Build Cache**. Avoid unnecessarily running components of builds and tests when inputs have not changed.

② **Predictive Test Selection**. Run only tests that are likely to provide useful feedback using machine learning.

③ **Test Distribution**. Run the necessary and relevant remaining tests in parallel to minimize build time.

④ **Performance Continuity**. Sustain Test Distribution and other performance improvements over time with data analytic and performance profiling capabilities.

[8] https://gradle.com/gradle-enterprise-solutions/test-distribution/#advantages-section

no parallelism - single machine parallelism(자원 공유 문제) - CI Fanout - Modern test distribution(TD)

CI Fanout

build 를 여러 CI jobs 로 나누는 것. manually configured 되어야 함. 그리고 CI platform 마다 unique 하게 할당이 되어야 함.

Test Distribution

특히 tests 가 JVM ecosystem 상에서 빌드 시간을 느리게 만드는 가장 큰 원인임.
gradle 상용화. 테스트를 나누어서 수행함.
로컬, ci 모두 가능.

General distribution

challenges : 환경 격리 + overhead time
bazel, pants

challenges : 어떻게 빌드를 split 할 것인지
언어에 따라서 성능 차이가 클 수 있음.

Common Factors of Remote and Distributed Builds

Network Traffic : serializing, Network proximity (hosts, storage)
Managing the pool of remote hosts or distributed agents

bazel 과의 비교 글들

gradle 이 bazel 과 비교해놓은 글들이 있음.[9] [10]
TODO 읽기.

Gradle work

어떻게 돌아가는지에 대한 글임. [11]
TODO 읽기. 3편.

---

[9] https://blog.gradle.org/our-approach-to-faster-compilation
[10] https://blog.gradle.org/gradle-vs-bazel-jvm#the-granularity-of-build-files
[11] https://blog.gradle.org/how-gradle-works-3