Build in the Cloud: Accessing Source Code

This is the first in a four part series describing how we use the cloud to scale building and testing of software at Google. This series elaborates on a presentation [http://www.youtube.com/watch?v=b52aXZ2yi08] given during the Pre-GTAC 2010 [http://sites.gtac.biz/gtac2010/] event in Hyderabad. Please see our first post [http://google-engtools.blogspot.com/2011/05/welcome-to-google-engineering-tools.html] for more details on the types of problems we are solving in Engineering Tools at Google.

Much of our day-to-day activities as software engineers involves source code. When we join a project one of the first things we do is look at the source. We want to build it, run it, experiment with changes, test it, and challenge our assumptions about how it works. For most of us this means we start by “checking out” the source from version control. For small to moderately sized projects almost any reasonable version control system is adequate. But as the number of engineers increases and the code base grows, this can put a strain on the version control system and decrease engineer productivity.

Here at Google, all products are built from head [http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html] . This approach has advantages: the code is open for anyone to explore and tinker with, it avoids the headaches associated with merging long-lived branches, and building from source ensures there are no binary compatibility issues between libraries. The downside is, with over a hundred million lines of code, it takes a long time to check out. And Google is a global company, which means checkout times are amplified in distributed offices. By computing dependency graphs and using this information to limit the number of files checked out, we have been somewhat successful in reducing checkout time. However, computing dependencies also takes time, and even with this improvement things still took too long.

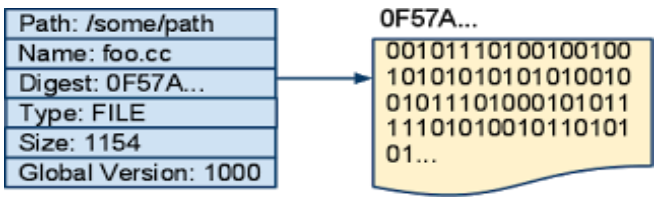
Engineer time lost to checking out code is the most obvious cost, but the true cost is much higher. Automated build and testing systems also need access to source code. Time spent checking

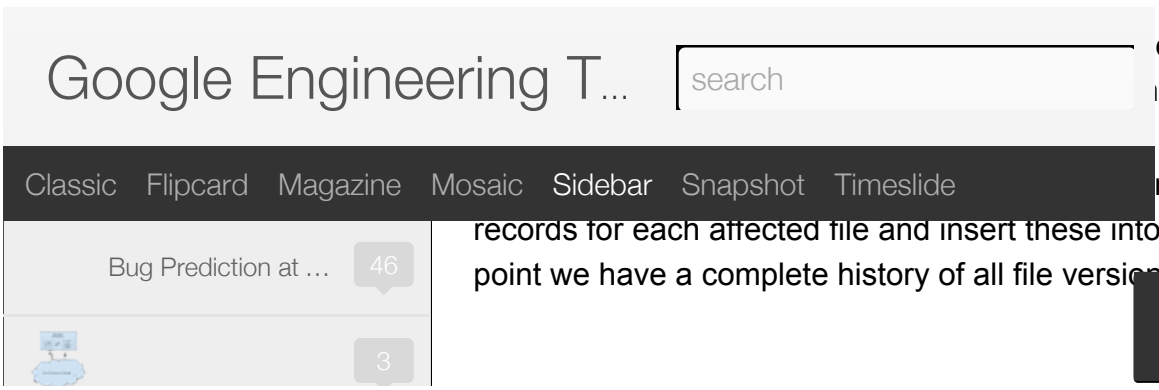
out code in these systems increases the feedback cycle, which decreases their utility. It also increases the complexity of these systems since they are required to maintain state on a file system and interact closely with the version control system for what is essentially read-only access to source code.

In fact, we have found that engineers check out and edit a very small amount code relative to the amount read to perform builds. This is because we always build from source, and changes tend to be localized to a small part of our source tree. So, both engineers and automated systems primarily need quick, read-only access to the large quantity of unedited code required to perform their builds. The unedited code itself is immutable, since it doesn't change once it's checked in to the version control system. This means we can use Google infrastructure to mirror all version control information in the cloud as a way to provide fast and scalable read-only access to source code.

In our system, every revision of every file has an associated metadata record. The metadata includes information such as the path location in the repository, the file name, size, version, and such. It also includes a digest of the content associated with this file revision. This digest is the hash of the file content itself, created using a hashing function appropriate for Content Addressable Storage [http://en.wikipedia.org/wiki/Content-addressable_storage] (CAS). There are many areas in our system where we utilize CAS; expect to hear more about it in subsequent posts.

A view of the metadata for version 5 of `/some/path/foo.cc` and CAS file content:





e version control
tents to compute
into BigTable
nstruct metadata
records for each affected file and insert these into BigTable. At this point we have a complete history of all file versions in the cloud.


Trying on the new… 1


Build in the Cloud: Distrib…


Build in the Cloud:… 12

Are SSDs a silver … 10


Build in the Cloud:… 8


Testing at the spe… 31

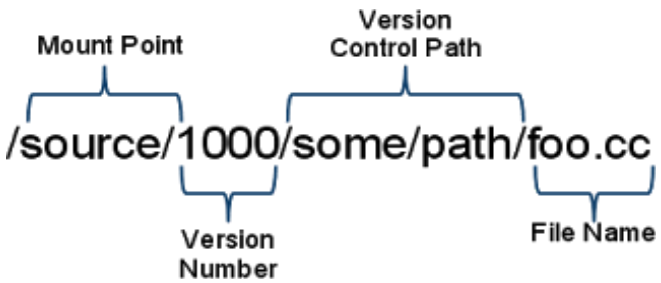

C++ at Google: H… 12

Welcome to the G… 10

Now that we have this data, what do we do with it? We could write a command line tool to checkout code from the cloud to the developer's workstation. This would be an improvement because it reduces load on the version control system, and replicating the data near distributed offices improves performance by reducing network latency. However it would be extremely useful if, instead of checking out source before it is actually needed, we automatically downloaded code on-demand.

This can be accomplished by writing a custom file system to provide a read-only view of the version history. The File System in Userspace [http://en.wikipedia.org/wiki/Filesystem_in_Userspace] (FUSE) kernel module is a convenient way of implementing a user space daemon that implements such a file system. Users interact with this source file system like any other file system, with the exception that certain path elements are special in that they configure the version information.

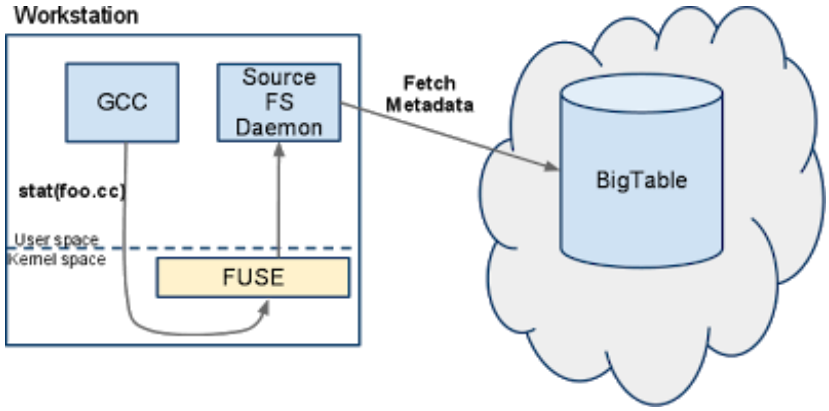
For example, accessing `foo.cc` at global version 1000 from our previous example:



[http://3.bp.blogspot.com/-SN4yL9oDk5Q/TgCqw6_VDFI/AAAAAAAAAAg/HzpC4CKe_jo/s1600/image01.png]

The file system resolves metadata calls such as `stat()` and `readdir()` by first interpreting the version number from the path and then querying the version control data in the cloud.

For example, using `gcc` (or any other tool) to compile `foo.cc`, where `gcc` first calls `stat()` to check file existence:



[http://1.bp.blogspot.com/-qZVDxPxJ8p0/TgCxNhh84LI/AAAAAAAAAAw/n41kZnan2fE/s1600/image00.png]

This allows users and tools to explore the version control using just the version metadata. File content is only downloaded at the point where it is opened. Since file versions are immutable, content can be cached and reused indefinitely. The use of CAS provides de-duplication of file content for free and the same content is never downloaded twice.

By putting our revision history in the cloud we provide engineers with complete access to all the source, and yet almost no time is spent checking out code. Replicating the cloud to multiple geographic regions ensures that performance is similar in all offices around the world. Finally, automated build and test systems can easily access code via a simple file system interface without dealing with the version control system directly. All of this adds up to a fast and efficient system which enables engineers and automated systems to focus on building and testing software instead of the mechanics of downloading source code.

Stay tuned! In this post we mentioned dependency analysis and Content Addressable Storage. Part two in this series describes our build system and how it relates to these topics.

- Nathan York

Posted 21st June 2011 by Engineering

8 View comments

Blair Kutzman June 27, 2011 at 1:03 PM

Super cool!

Sounds a lot like the 'views' from ClearCase.. But nice that is it

separate from the SCM system to avoid extra load and support multiple different systems at once.

Reply

Big 40wt Svetlyak June 28, 2011 at 1:01 AM

How fast is your FUSE implementation, does it caches files on a local hard drive?

How do you resolve security issues? Or all codebase is open to all employees regardles of their status?

Reply

nyork June 28, 2011 at 8:42 AM

We cache file content and metadata locally. The system overall is very fast for the way we use it and, most important, it scales. Some access patterns such as linearly scanning large numbers of files (ie grep -R) are not optimized since this use case is not common, and having the source in the cloud enables better alternatives.

Most of the code is accessible to engineers. We use the $USER information provided by the FUSE context for authorization purposes.

Reply

Providence Salumu July 13, 2011 at 1:15 AM

Nice presentation! Looking forward for the next ones...

Reply

dasch August 21, 2011 at 4:52 AM

I'd be interested to know which version control systems you primarily use within Google. The distributed nature of Git would make a good fit, but perhaps you use something more elaborate, or something home-cooked?

Reply

JamesB February 10, 2013 at 9:53 AM

Are there any plans to open source any of the implementation?

Reply

Ash August 26, 2013 at 3:18 PM

Can you elaborate how the "Global Version" from metadata is used? What I am not able to understand is, when you build, 1) How does the build system know which version of a file to use? or 2)For a build the version will be same for all the files? I don't think #2 is correct, because that would imply that all the files have changed.

Reply

alsaher qatar May 15, 2014 at 2:38 AM

Nice post.
Thanks for sharing valuable information.
Waiting for next ones…..

access control system

Reply

Enter your comment...

Comment as: Google Account

Publish Preview