Build in the Cloud: Distributing Build   puts

This is the fourth in a four part series describing   e 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. To get a sense of the scale and details on the types of problems we are solving in Engineering Tools at Google take a look at this post* [http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html] *.*

In the previous post, we saw how distributing build load across many machines and reusing the results of previous build actions can produce extremely fast build times. This distribution and reuse itself exposed other performance bottlenecks. Specifically, a full build of a large project may produce several gigabytes of output files, all of which need to be shipped from the cloud back to the developer's local machine. This taxes both our networks and the developer's local disk, the limited latencies and bandwidth of which slow down the build.

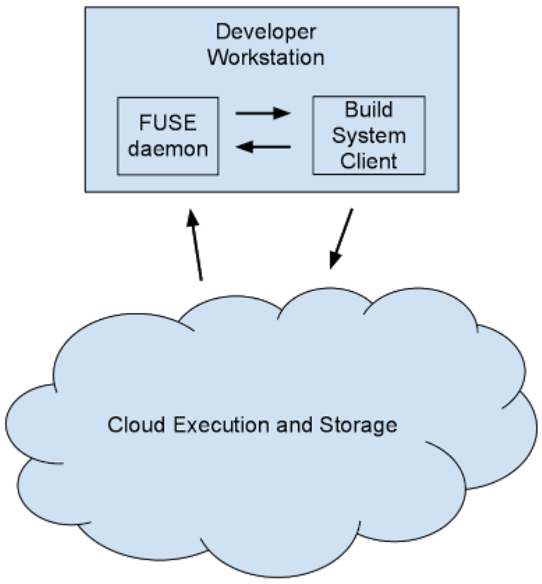
Furthermore, a developer typically does not need access to all of the outputs of a build. She may be interested only in the final binary that results from a build, not all of the intermediate object files. Another example is when the developer uses the distributed build system to build and execute a test in the cloud; in this case, she does not need local access to any build outputs at all, but simply an indication that the test passed or failed. It is thus unnecessary and wasteful to always ship all build outputs from the cloud back to the developer's local machine. Instead, it is sufficient to ship back only those build outputs that are actually accessed by the developer on demand.

To address these bottlenecks, our distributed build system writes all build outputs that it generates to a persistent, distributed storage system in the cloud. This storage system can handle sustained read and write rates many times those of a local disk, and the network connectivity between the distributed build system and this storage system has significantly lower latency and higher bandwidth than to the developer's local machine. The build outputs are stored by **content fingerprint**, providing a straightforward mechanism for indexing and retrieval of the files. (This cloud storage system also serves as a secondary caching layer for the distributed build system, providing an additional performance benefit in instances when build outputs fall out of the primary caching layer.) Using this distributed storage system in the cloud, we are able to make build speeds more than twice as fast as a distributed build that stores all build outputs on the developer's local disk.

With cloud-based storage of build outputs providing substantial increases in build times, the remaining challenge is to still make build outputs available to the developer's local machine, on demand, once the build completes. We achieve this by writing a custom file system to provide a view of the build outputs. Like we described in our first blog post [http://google-engtools.blogspot.com/2011/06/build-in-cloud-accessing-source-code.html] , 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. As individual build steps complete, the distributed build system informs the daemon about the creation of new build outputs, providing both a relative file path and a content fingerprint. Then, when the developer accesses one of the files on these paths, the daemon downloads the file from the cloud, using the content fingerprint as the key, and serves it via FUSE to the developer (caching downloaded files on the developer's machine to speed up subsequent accesses). The content fingerprint itself is also made available to the build system as an extended attribute via the FUSE layer, which, as we indicated in our second blog post [http://google-engtools.blogspot.com/2011/08/build-in-cloud-how-build-system-works.html] , is necessary for the build system to perform incremental builds.

In addition to serving build outputs from the cloud and build output metadata from the local machine, the local daemon performs many additional tasks important for the functionality of the system. For example, build outputs are kept in the cloud only as long as they are referenced by at least one developer's local machine. The daemon periodically extends leases for all build outputs present in any build on a developer's workstation. As another example, the performance of the virtual file system would be dismal if large build outputs were fetched sequentially once they were requested by the developer; instead, the daemon downloads chunks of the file in parallel from multiple storage servers in the cloud. The daemon is also responsible for local data integrity checks, local disk management and data eviction, and many other maintenance tasks.

The picture below summarizes how Google's build system works as a whole, taking into account the four blog posts in this series:



[http://4.bp.blogspot.com/-B0ZUuKTa5p4/Tqm42LG1NDI/AAAAAAAAABo/Qpfi41JG8XI/s1600/BuildintheCloudPart4DistributingBuildOutputs+%25282%2529.png]

The developer invokes the build system client on her workstation, specifying one or more targets to build. The client then coordinates and dispatches individual build actions to execute in the cloud, reading the necessary source code metadata from the FUSE daemon to provide as input to the cloud execution system. The build outputs of these actions (such as binaries) are then stored in our cloud storage system, and downloaded on demand by the user through FUSE daemon. The end result of this is a build system that can deliver build results to developers on the order of seconds or minutes, as opposed to hours.

This concludes our four-part series of blog posts describing how we use the cloud to scale building at Google. In this series, we covered cloud-based source control, how our build system works, and how we distribute both build actions and build outputs to the cloud. We encourage readers to comment on these posts, and to comment on any additional topics they would like to see covered in this blog.

- Milos Besta, Yevgeniy Miretskiy and Jeff Cox

Posted 28th October 2011 by Engineering

3 View comments

beelz December 21, 2011 at 7:20 PM

I had a few follow-up questions.

How does this translate to builds on non-Linux platforms (i.e.

Windows & Mac)? I'm guessing this is Linux/BSD only especially since the FUSE Mac driver was discontinued.

How do you get access to the command-line that was used to create a file? Does that mean that you also use custom Makefiles that all projects have to use? Or do you use something funky like reading the command-line of the process that opens the object file for writing?

Does this imply any requirements for the build system, tool chain, language, etc used for the project or is this fairly flexible? I.e. is this used for things like Java/GWT or is this mainly focused on C/C++ builds using gmake & the GNU toolchain?

Reply

mahima sharma September 2, 2013 at 11:38 PM

NET is a Programme Several important part of what Windows is running in Run On A Present For General klike program. **Cloud Development** This Cool People What do I need It. NET application that runs the computer That arrogant. For those who developed, .NET development, ASP development, SharePoint development, Microsoft development , software development, Singapore – Total eBiz Solutions Home

Reply

Biometric Attendance System Chennai March 19, 2014 at 5:47 AM

Hi,
Great information. As a person starting out, I find this information very useful, glad I stumbled upon your site. Great stuff, would love it if you created a whole course on this subject.

Fingerprint access control system Chennai

Reply

Enter your comment...

Comment as: Google Account

Publish Preview