SCons Remote Caching design

Introduction

Background

The SCons build system has support for a local cache in its CacheDir class, but no support for remote caching. At least one company that does all of its development in the Amazon cloud has overlaid the CacheDir class on top of a shared directory to provide distributed caching, but that is not a solution for distributed development teams.

This page is intended to provide a summary of our proposal for remote caching in SCons in the upstream code.

Hashing System

SCons currently only uses MD5 hashes. This document is mostly agnostic about the hash system, only discussing them when they impact the security or design of this feature. We intentionally use the term "content signature" (csig) as a synonym for hash.

Proposed Solution

We propose the following design:

1. Bazel remote cache server will be used as the action metadata and binary content store
    a. This provides a battle-tested solution that is known to scale.
2. Remote cache fetch (off by default) can be enabled as asynchronous communication
3. Remote cache push (off by default) can be enabled as asynchronous communication

This is accomplished using following high-level architecture in SCons:

1. A RemoteCache class in SCons's core layer owns communication with the Bazel remote cache server
2. SCons opts into a new scheduler from Job.py iff cache fetch is enabled
3. Together, these solutions ensure both CPUs and cache fetching are optimally scheduled/utilized

Important note: when remote caching is enabled, local directory-based caching via the CacheDir class will be disabled. It is possible to be able to support these together in the future, but that is not currently planned.

Technical Design

Bazel Remote Cache Server

Integration Design

The Bazel remote cache server is an existing server component that is designed to support `GET`, `HEAD`, and `PUT` requests for the following types of information:

1. Action metadata, stored under the `/ac/` section
2. Binary data, stored under the `/cas/` section

One important design principle for the Bazel remote cache server is that the signature of the action metadata stored under the `/ac/` section does not need to match the actual hash of the data, but the binary data under `/cas/` does need to match. Specifically, the Bazel remote cache will reject `PUT` requests under `/cas/` if the hash of the data that it receives does not match the hash provided in the URL. But it will not reject PUT requests under `/ac/` for that same reason.

Action Metadata

We will store one set of action metadata under the `/ac/` section per SCons Task object. This will allow us to do remote cache lookup on a per-Task basis. The action metadata will be formatted in JSON and will only contain a list of the targets for the task and any supporting data that we need, which is depends on the platform. All platforms will include the csig and size. Posix platforms will also include the mode. So for example, we could retrieve the following data for a task built on Windows with two targets:

```
{
    'build/subfolder/a.dll':
        {
            'csig': '<hash_of_file_contents>',
            'size': <file_size_in_bytes>
        },
    'build/subfolder/a.lib':
        {
            'csig': '<hash_of_file_contents>',
            'size': <file_size_in_bytes>
        }
}
```

or on Linux we could retrieve the following data for a task built with one target:

```
{
    'build/subfolder/a.so':
        {
            'csig': '<hash of file contents>',
            'size': <file size in bytes>,
            'mode': <st_mode value>
        }
}
```

We will generate the hash used in the `/ac/` request using the following code:

```
return SCons.Util.SignatureCollect(
    ['scons-metadata-version-%d' % self.metadata_version] +
    [t.get_cachedir_bsig() for t in task.targets])
```

The actual hash mechanism for SignatureCollect is TBD. See the Threat Modeling section at the end for more details about hash mechanism choices.

This metadata_version identifier will start at 1 and will be incremented every time we change the contents of the metadata. This allows us to change the metadata format without colliding with old consumers.

Changes Needed To Bazel Remote Cache Server

Currently the Bazel remote cache server only supports SHA-256 for requests (e.g. `GET http://bazel-cache.corp.int/cache/ac/<sha_256_hash>`), while SCons by default uses MD5. As part of this project, VMware will be contributing code to the upstream Bazel remote cache server project to support MD5 and SHA-1. We have received confirmation from the project maintainer that (1) it is acceptable to do this and (2) no prefix is needed for these alternative hashing formats. As a result, the requests SCons would make would be of the form `http://bazel-cache.corp.int/cache/ac/<md5_hash>` or `http://bazel-cache.corp.int/cache/ac/<sha1_hash>`. As mentioned before, see the Threat Modeling section at the end of this page for more discussion on hash formats.

SCons Integration

New Command-Line Flags

We propose the introduction of the following flags to SCons to control remote caching behavior:

1. --remote-cache-url
    a. URL of remote cache server
    b. Default: empty string
2. --remote-cache-fetch-enabled
    a. When True, fetches out-of-date tasks from remote cache before compilation
    b. Default: False
    c. Requires: --remote-cache-url
3. --remote-cache-push-enabled
    a. When True, pushes tasks to remote cache after compilation
    b. Default: False
    c. Requires: --remote-cache-url
4. --remote-cache-connections
    a. Specifies the number of threads that the RemoteCache class will maintain to dispatch asynchronous cache requests. Only applies if push and/or fetch is asynchronous.
    b. Default: 20

All parameters will also be settable via `SetOption` so that SConscripts can override dynamically.

RemoteCache Module

The RemoteCache module owns all network communication. It requires that the following modules be available:

1. urllib3
2. concurrent.futures (specifically ThreadPoolExecutor)

This class is responsible for doing remote cache push and fetch. It is entirely platform-agnostic and is intended to have no platform-specific code. The interface is as follows:

```
def raise_if_not_supported():
   """Raises an exception if the requires libraries are not available"""

class RemoteCache:
    def async_enabled(self):
        """Returns True if asynchronous fetch and/or push are enabled."""

    def set_fetch_response_queue(self, queue):
        """
        Sets the queue used to report cache fetch results if fetching
        is enabled.
        """

    def fetch_task(self, task):
        """
        Dispatches a request to a helper thread to fetch a task from the
        remote cache.

        Returns True if we submitted the task to the thread pool and False
        in all other cases.
        """


    def push_task(self, task):
        """
        Dispatches a request to a helper thread to push a task to the
        remote cache.
        """

    def close(self):
        """
        Releases any resources that this class acquired (e.g. the ThreadPool executor)

        Note: This function may be removed if we move ownership of the ThreadPool to
        the consumer for testability purposes
        """
```

New Job Scheduler For Fetches

As mentioned earlier, we will introduce a new job scheduler to be used when fetches are enabled. This scheduler is needed for two main reasons:

1. The existing Parallel scheduler is not efficient because it waits on jobs to be done when it should be fetching more tasks (see SCons pull request 3386 for more discussion)
2. The existing Parallel scheduler manages only one Queue, the job queue, while the new scheduler will need to manage two Queues: the job queue and the cache fetch result queue

The goal of this scheduler is to most efficiently manage the following three responsibilities:

1. Scanning for ready tasks using taskmaster.next_task()
2. Processing asynchronous cache fetch results
3. Processing action-related job results

The existing Parallel scanner did steps #1 and #3 above, but wasted time waiting on #3. This new scanner is expected to be more efficient because it only waits for queue results (steps #2 and #3 above) if it expects all other steps to already be exhausted. So for example, it would wait on cache fetch results only if the last run of taskmaster.next_task() returned None and we have no active jobs. Or as another example, it would wait on action-related job results only if the last run of taskmaster.next_task() returned None and we have no active asynchronous cache fetches.

The behavior of this job scheduler can be summarized using the following pseudocode:

```
class ParallelRemoteCache(Parallel):
    def start(self):
        while True:
            if we expect to have tasks left to scan:
                get the next task to execute
                if task is found and it is out of date:
                    try to fetch task asynchronously using the RemoteCache class

            if we don't have a task, there are no active jobs, and there are no pending cache fetches:
                build is done!

            for each pending job that is done:
                process the result of the job

            for each cache fetch that is done:
                if it is a cache miss:
                    send task to job scheduler thread pool
                else:
                    mark task as executed
```

Threat Modeling

SCons MD5 Hash Usage and Man-In-The-Middle Attacks

SCons MD5 hashes are not cryptographically secure, so attention must be paid to areas in which there are real attack surfaces. For threat modeling purposes, the Bazel remote cache assumes the following:

1. Entities you may not trust (e.g. your coworkers) can perform `GET` and `HEAD` requests to the cache server
2. Only trusted entities (e.g. official build farms, Jenkins, or Travis) can perform `PUT` requests to the cache server

These assumptions make us less concerned about cache poisoning. However, the Bazel remote cache can be configured to use http, so there is a legitimate attack surface related to man-in-the-middle attacks.

The Bazel remote cache requires that /ac/ and /cas/ requests use the same scheme, so unfortunately we can't do https requests to /ac/ and http requests to /cas/. With that limitation in mind, I believe that SCons MD5 hash usage does not open us up to any new attack surfaces. That is:

- If an organization uses a http transport, a man-in-the-middle can modify /ac/ results and there is nothing that we can do to prevent it.
- If an organization uses an https transport, a man-in-the-middle cannot modify any results so we do not worry about intentional nefarious hash collision.