Distributing Software in a Massively Parallel Environment

LISA 2014

Dinah McNutt
Release Engineer, Google, Inc.
November 12, 2014

Problem: Reliably and consistently distributing software in a Laaaaaaaaaaaarge Network

- Bottlenecks
  - Network
  - Disk
  - CPU
  - Memory
- Latency
- Machines Off-line
- Partitioned Networks
- Concurrent Writers

Solution: Google's Package Manager

- Midas Package Manager (aka MPM)
- Package metadata are stored in Google's Bigtable Database
- Package data are stored in Google's Colossus File System and are replicated
- Transport mechanism uses custom peer to peer mechanism (based on torrent)

Nope, this talk does not apply to Android

MPM Package Characteristics

- Contents (e.g. files)
- Unique version ID (a secure hash)
- Signatures
  - packages may be signed for verification and auditing purposes
- Labels
  - canary
  - live
  - release_candidate_2013_06_24_00
  - production=2013_06_24_00
- Pre-packaging commands
- Optional pre- and post-installation commands

Case Study: Project Xyzzy

- Configuration files need to be distributed to thousands of machines
- Files are packaged into an MPM
- Jobs on remote systems fetch a new version of the package every 10 minutes
- Post-fetch script within the package installs the configuration files

Sounds easy, right? Let's look under the covers...

Assumptions and Constraints

- There will always be off-line machines
- Bottlenecks need to be minimized
- Jobs must be able to specify which version of a package to use
- Jobs on same machine may use different versions of the same package
- Must be able to guarantee files have not been tampered with
- Must be able to rollback to a previous version

Package Creation

- Package definition file
  - list of files are included in package
  - file ownership and permissions
  - post-install and pre-deinstall commands
  - generated from build system
- Generated by build system
  - run build command
  - optionally apply label(s) and/or signatures to package
    - can be done now and/or later

Package Creation, cont.

- Package creation is idempotent
  - new package is not created if contents have not changed
  - transparent to user
  - requested labels get applied to the existing package

MPM Metadata

- Immutable
  - Who, when, how package was built
  - List of files, checksums, and file attributes
  - Some labels
    - production=2013_06_24_00
  - Version ID
- Mutable
  - Labels
  - Cleanup policy

Durability of Packages

Garbage collection performed automatically based on durability:

- Test
  - Reduced retention (3 days) and replication policies
- Ephemeral
  - Short-lived (7 days)
  - Useful for configuration files that are being packaged and pushed frequently
  - ***Saves on storage resources*** - packages are not kept longer than needed
- Durable
  - Default retention policy is 3 months since last used

Package Distribution

- Uses Pull method
- Pros:
  - Avoid network congestion - packages are only fetched when needed
  - Job owners can decide when to accept new versions of packages
- Cons:
  - Job owners can decide when to accept new versions of packages
  - Need extra logic in job to check for new version of packages OR the ability to re-start jobs easily
  - Can be difficult to tell who is going to be using a specific version

Metadata Distribution and Replication

- Metadata is stored in Bigtable (which is replicated)
- Root servers read and cache data from local Bigtable
- MPM client queries local root server
- Failover logic is in client
  - When MPM client requests fail, a request will automatically be sent to another root server (based on geographical location)

Package Data Distribution and Replication

- Package data is copied to centrally-managed replication servers
  - Stored in Colossus file system
  - Scattered geographically
  - 2-tiered architecture
    - frequently-used packages cached nearby
- Fetches are done using torrent-like protocol
  - Package data stored on local disk
  - Resilient to outages
- Very scalable
  - Millions of fetches daily
  - Petabytes of data daily

Security - Access Control Lists

- Package name space is hierarchical
  - storage/client
  - storage/client/config
  - storage/server
  - ACLs created on “storage” can be inherited by “storage/client” and “storage/server”
- 3 levels of access
  - owner - create and delete packages, modify labels, manage ACLs
  - builder - create packages and add/modify labels
  - label - controls who can add/modify specific labels
    - production.*
    - canary
    - my_immutable_label=dont_mess_with_me

Security - Encryption

- Individual files can be encrypted within a package
- ACLs define who can decrypt files
- Encryption and decryption performed locally (and automatically)
  - no MPM servers can read decrypted data

Security - Signatures

- Packages are signed at build time or later
- Secure key escrow service generates signature using:
  - package name
  - metadata (including file checksums)
- Signatures verified using the package name and expected signer
  - signed metadata re-verified against package contents

Re-visiting Project Xyzzy

- Package is created using MPM build command
  - If config files have not changed, a new package is not created
  - Package is ephemeral, no labels applied
- Metadata is written to Bigtable via API and automatically replicated
- Package is copied to Colossus and replicated
  - There may be a delay during replication
  - Information about the package is available immediately
  - Fetches may be performed from alternate Colossus server if replication is not complete (transparent to user)
- For project Xyzzy, a cron-like job will fetch the new version of package:
  - If package has not changed, nothing happens
  - If package data is already cached locally, it is not re-fetched

Cool Things I like about MPM

mpmdiff

- Can compare **any** two packages (don't have to have the same name)
- Can selectively report on most package attributes
  - file owner and/or mode
  - file size
  - post-install and pre-deinstall scripts
  - file checksums

Labels

- Can fetch packages using labels
- Can use labels to indicate where a package is in the release process (dev, canary, production)
- Packages can be promoted by moving labels from one package to another (labels are unique for a specific package)
- Some labels are immutable and cannot be moved
  - production_build=RC2013_01_12
- Some labels are special and cannot be specified by users
  - latest
- Labels can be used to assist in rollbacks
  - apply “rollback” or “last_known_good” label to current MPM when promoting the new one
  - SREs can quickly install the rollback MPM if there are problems

filegroups

- Files within an MPM can be stored in filegroups
- Filegroups can be fetched individually
- A file may exist in more than one filegroup
- Common practice is to store both stripped and unstripped binaries in same MPM but in different filegroups
  - This ensures that the unstripped binary matches the stripped version when troubleshooting problems

Web Interface

- Can browse all MPMs
- Displays Metadata
  - Version ID
  - Data and time package was built
  - Builder
  - Size
  - Labels
  - Date of last fetch
  - Durability
  - Expiration
  - ACLs
- Graphs showing package size (by filegroup) over time

Thank You

Questions?

Dinah McNutt
Release Engineer