Git Cache Maintenance Projects Idea

Brainstorming Together About Ideas and Alternatives

Recording: https://youtu.be/vg8SQEMSvks
Recording: https://youtu.be/Hmpe43bpvGY [April 1, 2022]

Objective

Meet for 60 minutes with those interested in the git caching project idea to discuss ideas and alternatives and to identify areas where there may be questions. Encourage discussion of different alternatives and ideas that might lead us to a better implementation.

User Experience

Because the git caches are stored on the Jenkins controller and are “system-wide”, it seems like the user interface to the caches needs to be accessed through a sub-page of the “Manage Jenkins” page. The concepts involved seem distinct enough that it will probably need its own dedicated page (similar to the dedicated page that is provided for plugins like the Configuration as Code plugin, the Implied Labels plugin, the Config File Provider plugin, and others)

Administrator Actions

Some of the actions that I believe administrators will want to perform related to the git caches that are on their controller include:

- Global configuration
    - How is the git executable chosen to perform the maintenance tasks?
        - Select it as a global configuration setting on the page?
            - Expect all tasks on the page to use git tool
        - If JGit is selected, then the controller process memory footprint will increase while JGit is performing the operation inside the controller JVM
        - If CLI git is selected, then a separate process is run and the memory footprint is not inside the controller JVM
    - Are there cases where it would be useful or helpful to use both JGit and CLI git?
        - What if JGit is significantly faster at some operation?
        - What if CLI git is significantly faster at some operation?
        - Performance optimization is usually a late stage, after the implementation is working and delivered
            - Would generally delay this optimization until evidence justifies it
    - Default tasks to be performed
        - Some tasks are correlated to size of repository
            - The gc operation may take a long time on a large repository

  - What mechanisms can we give the user to provide finer grained control of the maintenance tasks?
    - Override settings based on repository size?
    - Call a user provided shell script?
    - Call a user provided groovy script?
    - Repository based exclusion
      - If the repo URL is git://kernel.org/linux, only gc once a month
  - Should we prevent users from doing certain tasks?
    - Mark's thought was to not limit it
- Which tasks (of commit-graph, prefetch, gc, loose-objects, incremental-repack, pack-refs) are performed on all repositories by default and how would we select the tasks?
  - Task selection priority
    - Mark thinks prefetch has the opportunity to improve
      - Network traffic reduction is a likely win
      - Hourly in the git maintenance default
    - Man page review indicates we should run loose-objects
      - Daily in the git maintenance default
    - Hrushikesh recommends incremental-repack after prefetch
      - Daily in the git maintenance default
      - Optimizes access by doing the repack
      - See the stackoverflow article for incremental repack
    - The pack-refs task is not part of incremental strategy
      - Assumed not part of our default set
      - Can measure the impact of the pack-refs task on operations
    - Recommend
      - Task prefetch to retrieve new objects
      - Task loose-objects to copy loose objects to a pack
      - Task incremental-repack to combine smaller packs into larger packs
      - Task commit-graph to update commit-graph file
        - Updated incrementally in multiple files
        - Not an expensive operation
      - Task gc less frequently because it is very expensive
    - Could the proposal benefit by showing comparison between alternate strategies for running the tasks and highlight the differences one strategy
      - A few sample repositories (jenkins-bugs, linux kernel, others as MW example)
        - See the git client plugin source code for the URLs of some example large repositories

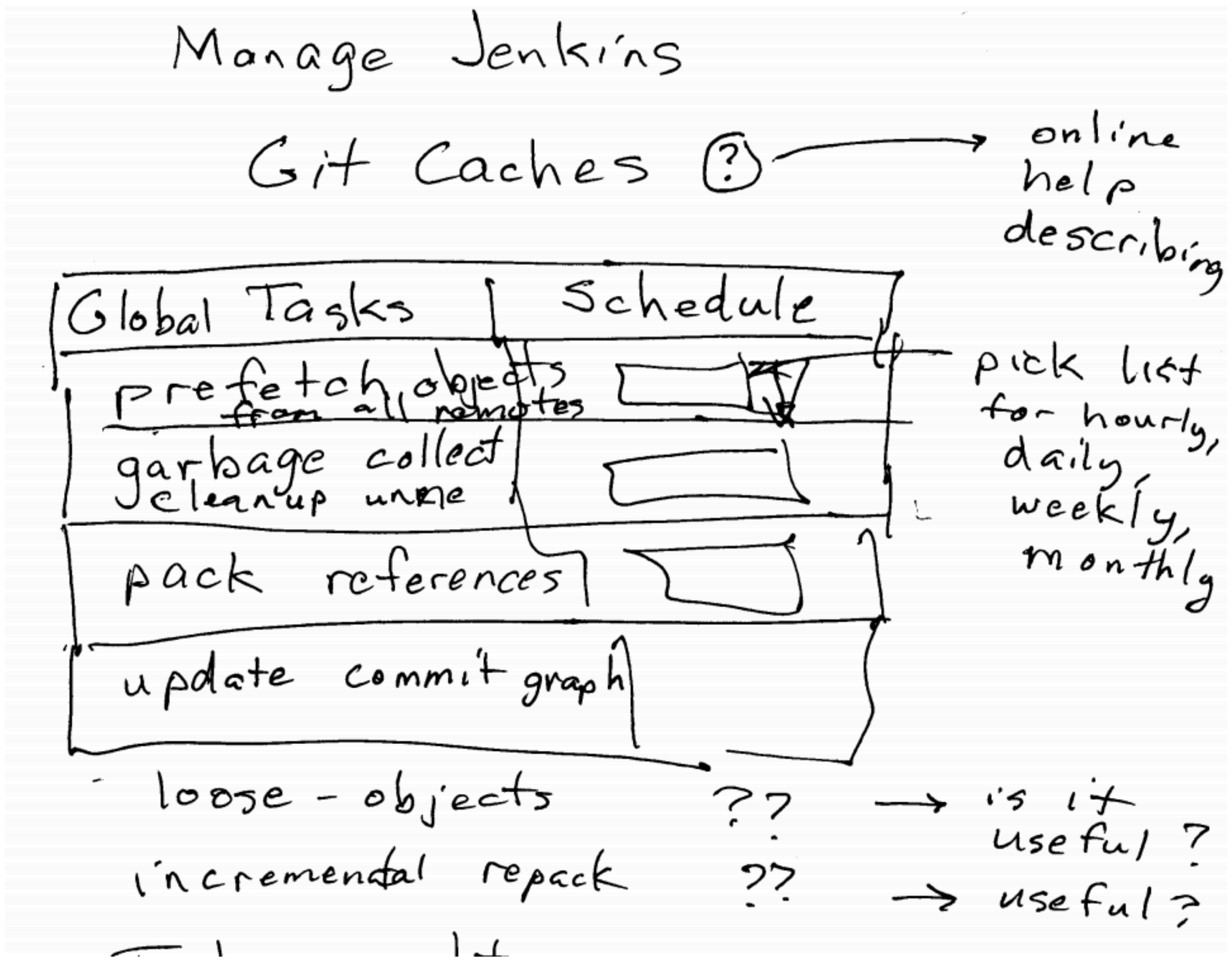
            - Run a series of comparisons of the operations and their impact on the repository
                - Bunch of new commits arrive
                - Prefetch - what's the impact?
                - Loose-objects - what's the impact?
                - Incremental-repack - what's the impact?
        - Need to assess the operations and their results based on the different command line git versions
            - Multi-pack-index requires git 2.20 or later
        - The commit-graph is updated at gc and then fetch updates the graph
            - Older versions don't default to update the commit-graph
        - Failure to update the commit-graph could slow a fetch
            - How would we measure that to decide?
            - Author of the blog (and the feature) notes that the fetch may be more expensive than if the commit graph had been updated by fetch
    - How do we decide the scheduling of the maintenance tasks?
        - Can we schedule maintenance tasks in parallel?
            - Our first priority is to "do no harm" - don't harm the controller with these tasks
            - Prefer serial tasks until proven otherwise
            - An administrator has configured gc to run every hour
            - Do we have a way to interrupt a maintenance task?
                - Serial execution may be easier to safely interrupt
        - What if that overloads the controller?
        - How do we allow the administrator to choose?
        - How do we measure and report the impact of maintenance?
    - Preferred development technique
        - Mark has a strong bias towards test driven development
            - Write tests as you go
            - Use the tests to explore
            - Learn from the tests
            - Don't be afraid to discard tests when they are not helping
        - Git plugin and git client plugin are difficult to test
            - Initially created without tests in first 18 months
        - 90 000 installations are using a new release of git plugin within the first 30 days after its release
    - See https://issues.jenkins.io/browse/JENKINS-13493 for discussion of the inspiration for this project idea
- Mark schedule recurring sessions immediately after Asia GSoC office hours
    - Weekly meet to discuss - Friday, April 1, 2022 3:30 AM UTC (9:00 IST)
- Schedule for tasks to be performed
    - How frequently are the tasks performed

    - Impact assessment of most recent tasks
        - How do we show the user that one or more repositories are causing an unexpected burden on their Jenkins controller?
            - For example, if they are caching the Linux kernel repository (over 2GB size repository), how will they be shown the duration of garbage collection operations or of prefetch operations?
        - Should we show improvements that maintenance tasks have provided?
            - For example, repository size before maintenance vs. after maintenance or number of objects retrieved by a prefetch
        - Should we offer recommendations for maintenance frequency based on patterns that we observe in the data?
            - If prefetch is performed hourly but only commonly finds changes once a week, should we recommend they adjust prefetch to be only daily?
        - How can we show the user the impact of the maintenance tasks on their controller?
            - Are there graphs or charts that will make the impact clear?
- Repository management
    - Register one or more caches to be automatically maintained (should we default to automatically register all caches?)
    - Unregister one or more caches to no longer be automatically maintained
- Task management
    - Run one or more maintenance tasks on one or more caches immediately
        - Tasks include commit-graph, prefetch, gc, loose-objects, incremental-repack, pack-refs
        - Should some of those tasks be excluded from the user interface because they are more obscure than most administrators will want to know?
        - Should we have an option in the user interface that passes the `--auto` argument to the job for an immediate run so that the operation is skipped if the cache does not need the operation that was requested
    - Schedule the frequency of maintenance tasks
    - View status of maintenance tasks
        - Tasks currently running
        - Task duration
        - Task overlap warnings (a task was still running when the next scheduled run of the task was started)
        - Task log
        - Failed tasks
    - Compare with a previous run of a maintenance task?
        - Changing configuration values of a maintenance task for a specific repo
            - For example, a very large git repo may need different options for the `git gc` command, allow those to be configured
- Permissions for the maintenance task configuration
    - Assuming the page is only available to a Jenkins administrator

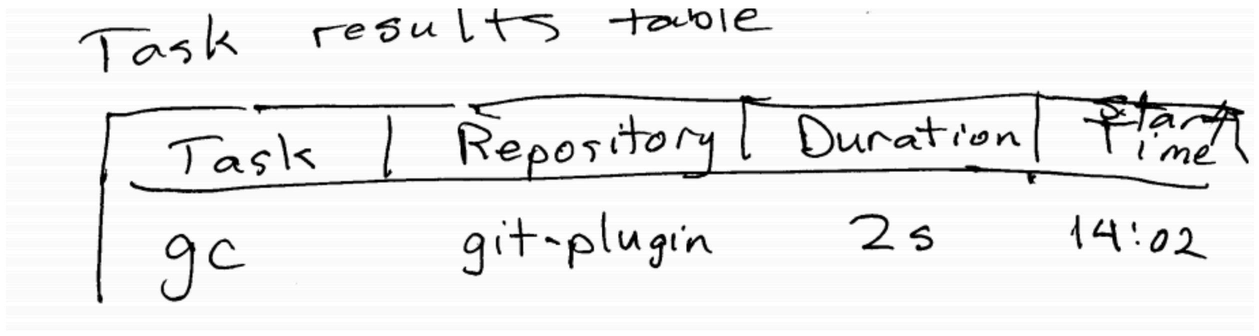
        - How do we show the maintenance task results in that context?
        - Mark assumes the maintenance tasks are not Jenkins jobs, not visible to the typical user
        - Mark assumes they may need to be lightweight tasks with history
    - Since tasks are a new concept, how does the administrator see the results of those tasks and configure those tasks?
        - Good for the proposal to describe how the administrator will interact with the
        - Mark assumes that cache maintenance is an administrator operation

Git Caches Page for Administrator (idea for discussion)

Here is a first scribbled sketch of a page idea for the top level page



I thought that the empty space at the bottom of that page could be used for a table of task results with links that allow the administrator to explore task results in more detail. For example:



Task results table

Task	Repository	Duration	Start time
gc	git-plugin	2s	14:02

If the columns of the task results table could be sorted, that would allow the administrator to identify the tasks that are taking a long time.

Seems like we'll also need some way to delete historical records from the task results table so that we don't use too much memory or disc for the results.

I think we might want a link to a trend chart included on each row that would show the historical trend for the task and repository represented by that row. Something like the trend chart that is shown for the build history of a job.

Implementation Issues and Concerns

This section of the brainstorming session will discuss areas of concern that we can predict will need to be considered during the implementation. This list is not a complete list, only a list of some of the items that we think could be encountered.

Wide Range of Git Implementations

The git plugin supports a wide range of command line git implementations, from command line git 1.8 on CentOS 7 to command line git 2.35.1 on FreeBSD and Windows and JGit 5.13 on all platforms. Some of those implementations do not implement key features that help with repository maintenance. For example, multi-pack indexes were first added to git 2.21. Attempting to perform a multi-pack index operation on an older version of command line git will fail with an error message. Others may provide an incomplete or flawed implementation of a feature and may need to be excluded.

The implementation will need to avoid calls to unsupported maintenance operations based on the version of command line git installed on the controller.

JGit also needs to be considered as part of the wide range of git implementations because JGit 5.13 may be missing features that are available in more recent versions of command line git.

User interface components

The Jenkins user interface is receiving significant improvements through the work of Jan Faracik, Tim Jacomb, Alexander Brandes, Ullrich Hafner, Daniel Beck, Adrien Lecharpentier, and others in the User Experience SIG. What can we do to assure that the new user interface additions in the git plugin cache management project are using the preferred user interface techniques?

The mentors don't know the preferred techniques. We may need to enlist help from others to recommend good techniques. Ullrich Hafner has spent significant time finding effective ways to present tabular and graphical information for the warnings-ng plugin. We may want to review ideas with him. Jan Faracik and Tim Jacomb are deeply involved in the UI rework and may be willing to guide UI component selection.

The ui-sample plugin is being actively improved now to show preferred techniques. See the UX SIG gitter channel for more discussions of the work on that plugin.

Localization of messages

The git plugin and git client plugin have not consistently applied localization techniques to messages. Should new additions be made ready for localization or should they continue the existing pattern?