Contributing to DistributedLog

This page documents the best way to make various types of contribution to Apache DistributedLog, including what is required before submitting a code change.

There may be bugs or possible improvements to this page, so please help us improve it. Credit to the Spark project. We've borrow liberally from their documentation.

Contributing to DistributedLog doesn't just mean writing code. Helping new users on the mailing list, testing releases, and improving documentation are also welcome. In fact, proposing significant code change usually requires first gaining experience and credibility within the community by helping in other ways. This is also a guide to becoming an effective contributor.

Overview

DistributedLog uses:

- JIRA and Github Issue to track logical issues, including bugs, tasks and improvements.
- Mailing Lists for proposing, planning, discussing changes
    - Developer: distributedlog-dev@bookkeeper.apache.org
        - Subscribe
        - Archives
        - Unsubscribe
- Confluence for documentation
- Github pull requests to manage the reviews and merge of specific code changes

That is, Mailing Lists are used for discussion and proposals, JIRA(or Github issue) and Confluence are used to describe what should be fixed or changed and high-level approaches, and Pull Requests and Review Board describe how to implement the change in code.

Code Source

A mirror of the repository is hosted on GitHub.

Contributions

By Helping Other Users

Helping answer user questions on the distributedlog-user@bookkeeper.apache.org mailing list is the first way to contribute to DistributedLog. There are always many new DistributedLog users; taking a few minutes to help answer a question is very valuable community service.

Contributors should subscribe to this list and follow it in order to keep up to date on what's happening in DistributedLog. Answering questions is an excellent and visible way to help the community, which also demonstrates your expertise.

By Testing Releases

DistributedLog's release process is community-oriented, and members of the community can vote on new releases on the distributedlog-dev@bookkeeper.apache.org mailing list. DistributedLog users are encouraged to subscribe to this list to receive announcements (e.g the DistributedLog 0.4.0 release vote), and test their workloads on the newer release and provide feedback on any performance or correctness issues found in the newer release.

By Reviewing Changes

Changes to DistributedLog source code are proposed, reviewed and committed via Pull Request (described later). Anyone can view and comment on active changes here. Reviewing others' changes is a good way to learn how the change process works and again exposure to activity in various parts of the code. You can help by reviewing the changes and asking questions or pointing out issues – as simple as typos or small issues of style.

Documentation Changes

To have us add a link to an external tutorial you wrote, simply email to distributedlog-dev@bookkeeper.apache.org.

To modify the built-in documentation, checkout the instructions on How to build website and documentation.

Preparing to Contribute Code Changes

Before proceeding to code changes, contributors should evaluate if the proposed change is likely to be relevant, new and actionable:

- Is it clear that code must change? Proposing a JIRA(or Github issue) and pull request is appropriate only when a clear problem or change has been identified. If simply having trouble using DistributedLog, use the mailing list first, rather than considering filing a JIRA(or Github issue) or proposing a change. When in doubt, email distributedlog-user@bookkeeper.apache.org first about the possible change.
- Search distributedlog-user@bookkeeper.apache.org and distributedlog-dev@bookkeeper.apache.org mailing list archives for related discussions. Often, the problem has been discussed before, with a resolution that doesn't require a code change, or recording what kind of changes will not be accepted as a resolution.
- Search JIRA / Github issue for existing issues.
  For JIRA, type "DistributedLog [search terms]" at the top right search box; while for Github issue, type "search terms" in the "Filters" search box.
  If a logically similar issue already exists, then contribute to the discussion on the existing JIRA/Github issue and pull request first, instead of creating a new one.
- Is the scope of the change matched to the contributor's level of experience? Anyone is qualified to suggest a typo fix, but refactoring recovery logic requires much more understanding of DistributedLog. Some changes require building up experience first (see above).

Code Review Criteria

Before considering how to contribute code, it's useful to understand how code is reviewed, and why changes may be rejected. Simply put, changes that have many or large positives, and few negative effects or risks, are much more likely to be merged, and merged quickly. Risky and less valuable changes are very unlikely to be merged, and may be rejected outright rather than receive iterations of review.

Positives

- Fixes the root cause of a bug in existing functionality
- Adds functionality or fixes a problem needed by a large number of users
- Simple, targeted
- Has tests
- Reduces complexity and lines of code
- Change has already been discussed and is known to committers

Negatives

- Adds complexity that only helps a niche use case
- Changes a public API or semantics (rarely allowed)
- Makes lots of modifications in one "big bang" change
- Band-aids a symptom of a bug only

Contributing Code Changes

Please review the preceding section before proposing a code change. This section documents how to do so.

When contributing code, you affirm that the contribution is your original work and that you license the work to the project under the project's open source license. Whether or not you state this explicitly, by submitting any copyrighted material via pull request, email, review board or other means you agree to license the material under the project's open source license and warrant that you have the legal authority to do so.

Open JIRA ticket / Github issue

1. Find the existing DistributedLog JIRA ticket and Github issue that the change pertains to.
   a. Do not create a new JIRA ticket / Github issue, if creating a change to address an existing ticket; add to the existing discussion and work instead.
   b. To avoid conflicts, assign the JIRA ticket to yourself(or comments on the github issue) if you plan to work on it.
   c. Look for existing pull requests or review boards that are linked from the the JIRA ticket(or github issue), to understand if someone is already working on it.
2. If the change is new, then it usually needs a new JIRA ticket / Github issue. However, trivial changes, where "what should change" is virtually the same as "how it should change" do not require a JIRA ticket. Example: "Fix typos in Foo javadoc"
3. If required, create a new JIRA ticket or Github issue:
   a. Provide a descriptive Title. "Update web UI" or "Problem in AsyncLogReader" is not sufficient. "Potential resource leak with unclosed AsyncLogReader" is good.
   b. Write a detailed description. For bug reports, this should ideally include a short reproduction of the problem. For new features, it may include a design document or a link to mailing list discussion.
   c. Set required fields:
      i. **Issue Type**. Generally, Bug, Improvement, New Feature, Documentation are the only types used in Distributedlog.
      ii. **Priority**. Set to Major or below; higher priorities are generally reserved for committers to set. JIRA tends to unfortunately conflate "size" and "importance" in its Priority field values. Their meaning is roughly:
         1. Blocker: pointless to release without this change as the release would be unusable to a large minority of users
         2. Critical: a large minority of users are missing important functionality without this, and/or a workaround is difficult
         3. Major: a small minority of users are missing important functionality without this, and there is a workaround

4. Minor: a niche use case is missing some support, but it does not affect usage or is easily worked around
5. Trivial: a nice-to-have change but unlikely to be any problem in practice otherwise

iii. **Component**

iv. **Affects Version**. For Bugs, assign at least one version that is known to exhibit the problem or need the change.

d. To avoid conflicts, assign the JIRA ticket to yourself(or comments on the github issue) if you plan to work on it. Leave it unassigned otherwise.

4. If the change is a large change, consider inviting discussion on the issue at distributedlog-dev@bookkeeper.apache.org first before proceeding to implement the change.

Git Workflow

Pull Request

1. Fork the Github repository at https://github.com/apache/incubator-distributedlog if you haven't already.
2. Clone your fork, create a new branch, push commits to the branch (review the DistributedLog Coding Guidelines , if you haven't already).
3. Consider whether documentation or tests need to be added or updated as part of the change, and add them as needed.
    a. Doc changes should be submitted along with code change.
    b. New server configuration settings should be added and documented in configuration file templates, and also documented in Documentation.
4. Run all tests with "mvn clean apache-rat:check package findbugs:check" to verify that the code compiles, passes tests and passes findbugs checks.
5. Open a pull request against the master branch of https://github.com/apache/incubator-distributedlog. (Only in special cases would the PR be opened against other branches.)
    a. The PR title should be of the form ***DL-XXXX: Title or ISSUE-XXXX: Title***, where *XXXX* is the relevant JIRA number or Github issue number, and *Title* may be the JIRA(or Github issue)'s title or a more specific title describing the PR itself.
    b. If the pull request is still a work in progress, and so is not ready to be merged, but needs to be pushed to Github to facilitate review, then add [WIP] after the JIRA(or Github issue)' id.
    c. Consider identifying committers or other contributors who have worked on the code being changed. Find the file(s) in GitHub and click "Blame" to see a line-by-line annotation of who changed the code last. You could add @username in the PR description to ping them immediately.
    d. Please state that the contribution is your original work and that you license the work to the project under the project's open source license.
6. A comment with information about the pull request will be added to the JIRA ticket.
7. Change the status of the JIRA to "**Patch Available**" if it is ready for review.
8. The GitHub CI hook will test your changes.
9. Once ready, the PR `checks` box will be updated with the test results along with a link to the full results on Jenkins.
10. Investigate and fix failures caused by the pull request.
    - Fixes can simply be pushed to the same branch from which you opened your pull request.
        - Jenkins will automatically re-test when new commits are pushed, if an existing commit is amended, if the branch is rebased or if the pull request is closed or reopened.
    - Despite our efforts, DistributedLog may have flaky tests at any given point, which may cause a build to fail. Unfortunately, it's not currently possible to restart the build by issuing a command. A workaround, as stated above, is to close and reopen the PR. If the failure is unrelated to your pull request and you have been able to run the tests locally successfully, please mention it in the pull request.

The Review Process

- Other reviewers, including committers, may comment on the changes and suggest modifications. Changes can be added by simply pushing more commits to the same branch.
- Lively, polite, rapid technical debate is encouraged from everyone in the community. The outcome may be a rejection of the entire change.
- Reviewers can indicate that a change looks suitable for merging with a comment such as: "I think the patch looks good. +1". DistributedLog uses "+1, -1" convention for indicating accepting or rejecting pull request, and also the pr merge script will use '+1' for pulling the reviewers' information..
- Sometimes, other changes will be merged which conflict with your pull request's changes. The PR can't be merged until the conflict is resolved. This can be resolved with "git fetch origin" followed by "git merge origin/master" and resolving the conflicts by hand, then pushing the result to your branch.
- Try to be responsive to the discussion rather than let days pass between replies.

Closing Your Pull Request / JIRA(or Github issue)

- If a change is accepted, it will be merged and the pull request will automatically be closed, along with the associated JIRA(or Github issue) if any
    - Note that in the rare case you are asked to open a pull request against a branch besides master, that you will actually have to close the pull request manually
    - The JIRA(or Github issue) will be Assigned to the primary contributor to the change as a way of giving credit. If the JIRA(or Github issue) isn't closed and/or Assigned promptly, comment on the JIRA(or Github issue).
- If your pull request is ultimately rejected, please close it promptly
    - ... because committers can't close PRs directly
    - Pull requests will be automatically closed by an automated process at Apache after about a week if a committer has made a comment like "mind closing this PR?" This means that the committer is specifically requesting that it be closed.
- If a pull request has gotten little or no attention, consider improving the description or the change itself and ping likely reviewers again after a few days. Consider proposing a change that's easier to include, like a smaller and/or less invasive change.
- If it has been reviewed but not taken up after weeks, after soliciting review from the most relevant reviewers, or, has met with neutral reactions, the outcome may be considered a "soft no". It is helpful to withdraw and close the PR in this case.
- If a pull request is closed because it is deemed not the right approach to resolve a JIRA(or close a Github issue), then leave the JIRA(or Github issue) open. However if the review makes it clear that the issue identified in the JIRA(or Github issue) is not going to be resolved by any pull request (not a problem, won't fix) then also resolve the JIRA(or close the Github issue).