HadoopContributorsMeeting20100528

0.21 release update

- Continuing to close blockers, ping people for updates and suggestions
- About 20 open blockers. Many are MapReduce documentation that may be

pushed. Speak up if 0.21 is missing anything substantive.

- Common/HDFS visibility and annotations are close to consensus;

MapReduce annotations are committed to trunk and the 0.21 branch

HEP proposal

Slides from the meeting: http://www.slideshare.net/cloudera/hadoop-contributors-meeting-ii

(what follows is the sketch presented at the meeting. A full proposal with concrete details will be circulated on the list)

- Based on- and very similar to- the PEP (Python Enhancement Proposal) Process
- Audience is HDFS and MapReduce; not necessarily adopted by other subprojects

- Addresses the perception that there is friction between

innovation/experimentation and stability

- Not for small enhancements, features, and bug fixes. This should not

slow down typical development or impede casual contribution to Hadoop

- Primary mechanism for new features, collecting input, documenting

design decisions

- JIRA is good for details, but not for deciding on wide shifts in direction
- Purpose is for author to build consensus and gather dissenting opinions.

- All may comment, but Editors will review incoming HEP material
- Editors determine only whether the HEP is complete, not whether

they believe it is a sound idea

- Editors are appointed by the PMC
- Mechanism for appointing Editors and term of service TBD
- Apache Board appoints Shepherds for projects somewhat randomly, to projects. A similar mechanism could work for incoming HEPs
- Proposal *may* come with code, but not necessarily. Drafting/baking of the HEP occurs in public on a list dedicated to that particular proposal. Once Editors certify the HEP as complete, it is sent to general@ for wider discussion.
- The discussion phase begins on general@. The mailing list exists to ensure the HEP is complete enough to present to the community.
- Some discussion on the difference between posting to general@ and posting to the HEP list. Completeness is, of course, subjective. If the Editor and Author disagree whether the proposal affects an aspect of the framework enough to merit special consideration, it is not entirely clear how to resolve the disagreement.
- In general, the role of the Editor in the community-driven process of Hadoop is not entirely clear. It may be possible to optimize it out.
- Once discussion ends, the HEP is passed (or fails to pass) by a vote of the PMC (mechanics undefined). In Python, the result is committed to the repository. A similar practice would make sense in Hadoop.

- Which issues require HEPs?

- Discussion ranged. Append, backup namenode, edit log rewrite, et

al. were examples of features substantial enough to merit a HEP. Pure Java CRC is an example of an enhancement that would not. Whether an explicit process must be in place to determine whether an issue requires a HEP is not clear.

- Viewing HEPs as a way of soliciting consensus for an approach might be more accurate. Going through the HEP process should always improve the chances of a successful proposal

- Evaluation

- The proposal may be rejected if it is redundant with existing

functionality, technically unsound, insufficiently motivated, no backwards compatibility story, etc.

- Implementation is not necessary, and is lightly discouraged. Feedback is less welcome once code is in hand.
- Purpose is to be clear about the acceptance criteria for that issue, e.g. concerns that the proposal may not scale or may harm performance
- Dissenting opinions must be recorded accurately. Quoting would be a safe practice for the Author to encourage HEP reviewers not to block the product of the proposal.

- The testing burden and completion strategy may be ambiguous

- Whether the proposal affects scalability may not be testable by

the implementer. Completing the proposal to address all use cases may require considerably more work than the Author is willing or motivated to invest.

- The HEP discussion on general@ should explore whether such objections are merited and reasonable. For example, a particularly obscure /esoteric use case could be included as a condition for acceptance if the dissenter is willing to invest the resources to test/validate it. The process is flexible in this regard.
- But it is not infinitely flexible. Backwards compatibility, performance regression, availability, and other considerations need not be called out in every HEP.
- Traditional concerns need to be documented. Acceptance criteria should ideally be automated and reproducible in different organizations

Branching

- A patch and a branch are isomorphic from a policy perspective. Of

course, they are functionally distinct: branches are easier to collaborate on and are, generally, longer-lived than are patches. But special policies need not be derived to account for these differences, which concern the production of the code, not its review and acceptance.

- Some developers find branches to be easier to review than very large

patches and easier to merge, given a toolchain that supports this.

- Subversion currently is difficult to adapt to this model
- Could be done on a HEP-by-HEP basis, as a condition for acceptance

- Eclipse Labs

- Branded version of Google Code (same functionality, w/ Eclipse brand)
- Not official Eclipse projects, but associated with Eclipse
- Apache/Hadoop may consider a similar strategy
- Distinct from Apache Labs, as one need not be a committer, follow

its rules for releases, etc.

Contrib

- Modules (such as fuse-dfs) are not actively maintained in the main

repository and would benefit from a release schedule decoupled from the rest of Hadoop

- With few exceptions, the contrib modules have smaller, often

discrete groups of maintainers. It may be worth exploring whether these projects could live elsewhere