Release1.0Requirements

Hadoop Release 1.0

This page is a place to collect potential requirements for a 1.0 release of Hadoop.

A jira has been created for this: https://issues.apache.org/jira/browse/HADOOP-5071

Is Hadoop Mature for 1.0 Release

This section is for a discussion on whether the Hadoop components are mature enough for a 1.0 label. If we are expecting a lot of flux in features that require changes to API, then 1.0 may be premature.

Another definition of 1.0 is that the product is "ready for users". Hadoop has long been ready for use, and indeed enjoys broad use. So, by this argument, we could declare 1.0 sooner, and then use 2.0 for our next, major, incompatible release. --DougCutting

Question: Does the release number really matter? Should we just keep adding features, improving back-compatibility, etc? Our Roadmap currently defines what a major release means. Does this need updating? – !DougCutting

What does Hadoop 1.0 include?

Question: Do we assume that all of these subprojects will go 1.0 at the same time? – !DougCutting

What does Hadoop 1.0 mean?

- Standard release numbering: Only bug fixes in 1.x.y releases and new features in 1.x.0 releases.
- No need for client recompilation when upgrading from 1.x to 1.y, where x <= y
  - Can't remove deprecated classes or methods until 2.0
- Old 1.x clients can connect to new 1.y servers, where x <= y
- New !FileSystem clients must be able to call old methods when talking to old servers. This generally will be done by having old methods continue to use old rpc methods. However, it is legal to have new implementations of old methods call new rpcs methods, as long as the library transparently handles the fallback case for old servers.

Owen, you seem to be extending the Roadmap's compatibility requirements to RPC protocols, is that right? Clients must be back-compatible with older servers and servers must be back-compatible with older clients. If so, perhaps we should vote on this new policy and update Roadmap accordingly. We could even start enforcing it before 1.0, so that, e.g., 0.20's protocols would need to be back-compatible with 0.19's but 0.21's would not. --DougCutting

Doug, This is a subtle issue: Here we are talking about new client application code calling old methods on the !FileSystem class (not the RPC methods). This is in addition to the rpc compatibility issue. --!SanjayRadia

Sanjay, How does 'new client application code calling old methods on the !FileSystem' differ from our current back-compatibility requirements? --DougCutting

_ Doug, currently when a new RPC method is added, we declare a change in protocol version and old and new can no long talk via rpc. There are two issues. One is the wire compatibility as you noted (this is new). Second is that when we add new rpc methods, the client side library can use the new rpc method but it must fall back on old rpc methods when talking to old servers (this must be explicitly coded in the library – it won't happen just by introducing wire compatibility). You are right in that is nothing new as far as API compatibility. Maybe this point is obvious to most, but Owen and I discussed it at some length and thought it was a subtle twist that our developers will have to be aware of in 1.0._

Sanjay, we should vote on this as a change to our back-compatibility policy, and, if it passes, start enforcing it. --DougCutting

I am not sure if we as community can start enforcing protocol compatibility right away because in order to do that we would have to version the rpc parameters. One we version the data types we can change the policy and enforce it. However we can easily support backward compatibility when new methods are added. I am not sure if this subset can be documented in the backward-compatibility policy (more correctly I don't know how to word it); however I will file a jira to change the rpc layer to detect and throw exceptions on missing methods. I will also follow that up with a patch for that jira. This will allow is us to treat the addition of a new method as a backward compatible changes that does not require the version # to be bumped.--SanjayRadia

A discussion on Hadoop 1.0 compatibility has been started in a email thread sent to *core-dev@hadoop.apache.org* with the subject *Hadoop 1.0 Compatibility Discussion* - please read and comment there.

Time frame for 1.0 to 2.0

What is the expectation for life of 1.0 before it goes to 2.0 Clearly if we switch from 1.0 to 2.0 in 3 months the compatibility benefit of 1.0 does not deliver much value for Hadoop customers. A time frame of 12 months is probably the minimum.

The [Roadmap] suggests that major releases will be approximately annually. I wouldn't put 12-months as a hard minimum, but rather as an approximate goal. Generally, we're slower than our goals, so I'd be surprised if it was much less than a year. --DougCutting

Prerequisites for Hadoop 1.0

Please add requirements as sections below. If you comment on someone else's requirement, please add your name next to your comments.

New MapReduce APIs

This is https://issues.apache.org/jira/browse/HADOOP-1230. The new API will provide us with much better future-proof APIs. The current Map/Reduce interface needs to be removed.

HDFS and MapReduce split into separate projects

This was agreed to in a thread on general@. There will be three separate projects, mapred, hdfs, and a library of common APIs and utilities (fs, io, conf, etc. packages).

Multi-language serialization

- A serialization format that will allow non-java clients to be supported. Note 1.0 does not need to provide non-java clients but we should be able to support them before going to 2.0.

Do you mean serialization of user objects, or RPC parameters? We have support for multi-lingual user objects already. As for RPC, I don't yet see the case for forcing 1.0 to wait until we're ready to support multiple native client libraries. --DougCutting

I mean RPC parameters. This one is boarderline. We are hurting for not have a language neutral serialization. Language neutral serialization is not that hard to do. There are many schemes that do this (Sun RPC, Protocol Buffers, Etch, Hessian, Thrift). We have to pick rather than invent.

I think for 1.0 we should not switch Hadoop's RPC mechanism. That is a good change to target for 2.0, when the RPC landscape is clearer. Language-neutral protocols also mean duplicated client-side logic which brings considerable complication. --DougCutting

I wasn't suggesting changing the RPC protoocol but merely the layer above the rpc protocol - the rpc type system and/or serialization. For example, protocol buffers lets you use your own transport. So may thrift. I would also like to wait till a good rpc solution is available. However I would like to understand what you think is missing in the current RPC landscape? Are there specific features that you see missing in some of the rpc solutions or are you waiting for CISCO's Etch solution to make a decision. Etch looks very attractive on the surface. I am also torn by this decision because it seems worth waiting till Etch is published before making our decision. It would help if you could elaborate on your thoughts on the rpc issue. --SanjayRadia

What's missing from the current RPC landscape? Mostly transport-layer stuff. (1) Transport versioning. Thrift doesn't provide transport-level handshakes, so we'd probably need to implement our own transport. This is possible, and we'd have to do it for protocol buffers too, but we might not with Etch. (2) Async transport. For performance we need async servers at least, and probably async clients. Requests and responses must be multiplexed over shared connections. Thrift doesn't yet provide this for Java. Etch may solve both of these or none or have other problems. It would be nice to get as much as possible from an external project, reinventing the minimum. So we should certainly start experimenting now. Someone could, e.g., port Thrift and/or protocol buffers to run on top of Hadoop's existing transport layer. We could immediately incorporate any improvements that make the transport more easily usable for Thrift and Protocol Buffers, and we'd probably identify other issues in the process. Fundamentally, I don't think switching the RPC is a move we can schedule without more work up front. But we should certainly start experimenting now. --DougCutting

*RPC systems are usually 2 layers: a RPC type system (where the rpc interfaces, parameter types, and serialization are define) and a RPC transport. We should not pick a rpc system that does not allow one to slip in arbitrary transports. Hadoop rcp transport is fairly good now and in the very short term we should consider sticking to it. The additional transports features you have mentioned further argues for this separation and independence. So if there is a RPC system that offers a pluggable transports AND also has a suitable RPC type system, we can safely pick that RPC system. We stick with Hadoop transport initially and later consider extending it or switching to another (possibly the native transport of the rpc system). For example Protocol Buffers does not even offer a transport - you *have* to plug one in. Having said that I do share your view that it would be nice to wait for Etch to become public and see if it has all or most of what we want. What we can't afford to do is to wait too long ... there will always be another new rpc system just around the corner. Further if our goals are a prefect RPC type system and a perfect RPC transport as single package, then we risk waiting forever. Pick a good enough one and we can work to extend it. The key to succeeding in picking one (now or waiting and deciding in 6 months) is to focus on the choosing one that has a good rpc type system AND allows a pluggable transport. --SanjayRadia*

Language Neutral - yes it will mean duplicated client-side and hence more work. From what I have observed in the design discussion, keeping the client side small was a criteria because we were expecting language neutral protocols down the road. Do you feel that we should not bother with language neutral protocols at all? --SanjayRadia

I think we should be very careful about which network protocols we publicly expose. Currently we expose none. I do not think we should attempt to expose all soon. A first obvious candidate to expose might be the job submission protocol. Before we do so we should closely revisit its design, since it was not designed as an API with long-term, multi-language access in mind. Any logic that we can easily move server-side, we should, to minimize duplicated code. Etc. The HDFS protocols will require more scrutiny, since they involve more client side logic. It would be simpler if all of HDFS was implemented using RPC, not a mix of RPC and raw sockets. So we might decide to delay publicly exposing the HDFS protocols until we have made that switch, should it prove feasable. I think we could reasonably have a 1.0 release that exposed none. I do not see this as a gating issue for a 1.0 release. We could reasonably expose such protocols in the course of 1.x releases, no? --DougCutting

*I wasn't suggesting making the network protocols public - that would be a serious undertaking - much more than making an API public. Since HDFS protocols have non-trivial client side library, i don't expect anyone to use the protocol directly. I would not want to consider exposing of the HDFS protocol before 2.0 and maybe much much later. (I am not sure the mix of rpc and raw makes things much worse ...). (BTW If you recall we recently made DFSClient private - that was a very deliberate action). So I believe we are in agreement here. My motivation for language neutral was for direct access via C and a scripting language. I was expecting that we would supply the client side for these languages and that we would *not* expose the protocol. I see language neutral marshalling to be a small, first-step, towards a non-java client. --SanjayRadia*

Versioning Scheme - Manual or Automated

Hadoop is likely to see fairly significant changes between 1.0 and 2.0. Given the compatibility requirements, we need some scheme (manual or automated) for versioning the RPC interfaces and also for versioning the data types that are passed as parameters to rpc.

We already have manually maintained versions for protocols. Automated versions will make some things simpler (e.g., marshalling) but won't solve the harder back-compatibility problems. We could manually version data types independently of the protocols by adding a 'version' field to classes, as is done in Nutch (search for readFields method), but that method doesn't gracefully handle old code receiving new instances. A way to handle that is to similarly update the write() method to use a format compatible with the client's protocol version. Regardless of how we version RPC, we need to add tests against older versions. --DougCutting

A manual scheme is too messy and cumbersome. An automated scheme a la Protocol Buffers, Etch, Thrift, Hessian should be used. – SanjayRadia

For 1.0 we should not switch Hadoop's RPC mechanism. That is a good change to target for 2.0, when the RPC landscape is clearer. We have very specific performance needs for RPC that these other mechanisms do not yet support. -DougCutting

Sanjay, about versioning RPC parameters: On the mailing list I proposed a mechanism that, with a small change to only the RPC mechanism itself, we could start manually versioning parameters as they are modified. Under this proposal, existing parameters implementations would not need to be altered until they next change incompatibly. It's perhaps not the best long-term solution, but it would, if we wanted, permit us to start requiring back-compatible protocols soon. --DougCutting

_ Yes I saw that and am evaluating it in the context of Hadoop. I have done similar things in the past and so know that it does work. I will comment further in that email thread. Thanks for starting that thread. --SanjayRadia_

Doug has initiated a discussion on RPC versioning in a email thread sent to *core-dev@hadoop.apache.org* with the subject *RPC versioning* - please read and comment there.

RPC server that's forward-friendly

- RPC mechanism should be able to detect and throw exception on new methods that old servers clearly don't implement.

That could be added in 1.1, no? I don't see this as a 1.0 requirement, but maybe I'm missing something. What happens today when you call an undefined method? Protocol version mismatch, probably. --DougCutting

Strictly speaking, this can wait till 1.1 but we have to be sure that we can do it. Most rpc systems do this as part of its spec and not as an afterthought. Given that it is easy to do I would prefer to do it for 1.0 – SanjayRadia.

Release 2.0 Requirements

List here longer-range items, that we'd like to do someday, but not in the 1.0 timeframe.