Googlers: shared externally

Device (PJRT) API ABI versioning and compatibility

jieying@google.com, skyewm@google.com, phawkins@google.com, fishx@google.com

Status: Draft

Last update: 18 Sept 2023

Summary

This doc focuses on the ABI versioning and compatibility of Device (PJRT) API.

- Framework provides a N-week forwards compatibility window for minor version updates (forward compatibility = framework is newer than plugin)
    - The compatibility window starts when the change is committed. A changelog with dates will be added.
    - A release done at time T should support all the versions between (T - N weeks) and T.
    - As a start, N will be 6 weeks. It will be extended to 12 weeks in the future.
- The first major update will be between 6 - 12 months. Aim for updating the major version once every year afterwards. Major version update will be posted as a PR 4 weeks before the update. Frameworks do not support plugins with a lower major version.
- Plugins define their own backward compatibility policy, i.e. the behavior when the plugin has a higher version than the framework.

Versioning of Device API (i.e. PJRT C API)

A Major and a Minor version number are used for the versioning of Device API (i.e. PJRT C API). Please see the tables in the next section about what changes are major version changes and minor version changes respectively.

Compatibility of Device API (i.e. PJRT C API)

This section focuses on the ABI compatibility of the Device API (i.e. PJRT C API) itself, and does not include the compatibility of other sources such as StableHLO. Note the PJRT plugin is the integration point to return information for all information related to compatibility (through the plugin attributes query API). The framework can check the compatibility window for each potential incompatible source, and give a conclusion whether the plugin is compatible with the framework.

Minor version update

API change	Plugin	Framework
• Add a new method. • Add a new field to argument structs. • Rename a method or an argument or a field in the argument struct. • Deprecate a method.	The plugin needs to update its version within the compatibility window. Otherwise the version check in the framework before calling the plugin methods will fail.	A release done at time T should support all the versions between (T - compatibility window). We will start with 6 weeks as the compatibility window. It will be extended to 12 weeks in the future.

To support versions within the compatibility window, the framework will:

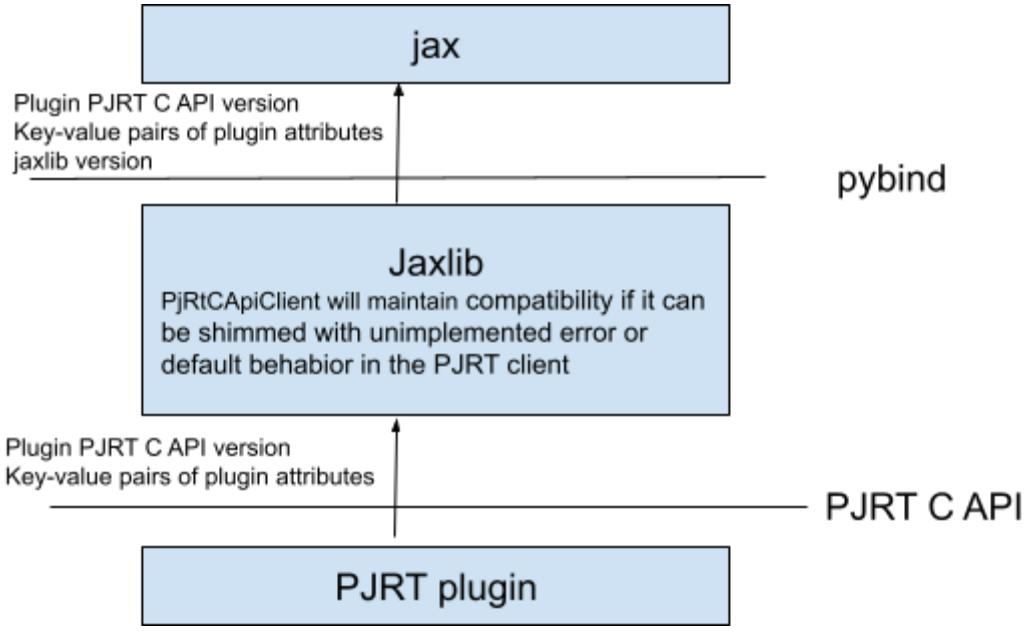
- Use the new method or the new field if the plugin has the same version as the framework.
- Otherwise the framework will return an unimplemented error or return a default result to support the M.N behavior.

```
C/C++
// Framework code
// Assume PJRT_Memory_Kind was added in PJRT_API_MINOR 14
if (pjrt_api->pjrt_api_version.minor_version >= 14) {
  return pjrt_api->PJRT_Memory_Kind(...);
} else {
  return unimplemented or default;
}

// Assume method PJRT_Client_A is renamed to PJRT_Client_B in MINOR 2.
pjrt_api->PJRT_Client_B(...);  // Will call PJRT_Client_A function ptr in
plugin
```

How JAX will handle minor version update

JAX is split into two libraries: jax (Python code) and jaxlib (C++ code). jaxlib calls the PJRT C API. JAX is already managing compatibility between jax and jaxlib, and compatibility between jaxlib and plugin is another layer of compatibility.



- PjRtCApiClient
  - Provide one method to get the major and minor PJRT C API version of the plugin. This method will be exposed to jax.
  - PjRtCApiClient will handle compatibility as the codeblock above: if plugin api version >= N: call new method; else return unimplemented or fail or returns a default value

```C/C++
// in pjrt_c_api_client.cc
class PjRtCApiClient : public PjRtClient {
 public:
  // Returns the major and minor PJRT C API version of the plugin.
  std::pair<int, int> api_version();

 private:
   int major_version_;
   int minor_version_;
}

// Assume layout was added to BufferFromHostBuffer in version N, and it is
within
// the compatibility window.
PjRtCApiClient::BufferFromHostBuffer(...) {
  BufferFromHostBuffer_Args args;
  if (minor_version_ > N) {
   args.device_layout = ...;
```

```
  }
  c_api_->PJRT_Client_BufferFromHostBuffer(args);
  ...
}

// py_client.h will be modified accordingly to add api_version() in xla.cc
py::class_<PyClient, std::shared_ptr<PyClient>> py_local_client(m, "Client");
py_local_client.def_property_readonly("api_version", &PyClient::api_version)
```

- Some decisions can not be shimmed by PjRtCApiClient, for example (1) different PjRtCApiClient methods need to be called depending on the plugin version, (2) in some case jax does not want to call multiple a series of methods if it knows they will all return unimplemented errors. In these scenarios, jax can make the decision by calling api_version. jax may need to check both api_version and jaxlib version depending on the feature. The minimum jaxlib version can be bumped without considering the compatibility window of PJRT C API changes.

```
Python
if (client.api_version[1] > N && xla_extension_version > K) {
 ...
} else {
 ...
}

if (client.attributes['support_memory_space']) {
 ...
} else {
}
```

Who's responsibility is it to implement forward compatibility
Whoever makes the PJRT C API changes will be responsible for implementing the forward compatibility, but jieying and skyewm can provide guidance if needed.

Major version update

The major version update will be done in a planned manner. The first major update will be between 6 - 12 months. We will aim for updating the major version once every year afterwards. Breaking changes in the major version update will be posted as a PR four weeks before the update happens.

API change	Plugin	Framework

• Deleting a method or argument • Changing the type of an argument • Rearranging fields in the PJRT_Api or argument structs	Plugin will need to coordinate and release for the new major version. Plugins with older major versions will not be supported.	The framework will have a release version that only supports the newer major version and does not support older versions.

Framework (e.g. JAX) has an older version than plugin (backwards compatibility)

If the framework releases less frequently than the plugin, or the plugin has releases for bug fixes, the framework may have an older version than the plugin. It is the plugin's responsibility to define the behavior when the framework has an older version (e.g. when the framework does not set certain new fields). The plugin can choose to abort or let the plugin behave as an older version.

Testing process

JAX provides a set of tests (e.g. existing python tests and some C API specific tests which will be added soon). Plugin can run these tests with plugin implementations and desired JAX version before plugin releases.

JAX can maintain a CPU/GPU/TPU plugin and it can run tests against its plugin at different PJRT C API versions. If the JAX release is done within a compatibility window of the earlier PJRT C API version, the release should pass testing with a plugin at a version that is within the compatibility window. For example, JAX already has a CI for Cloud TPU nightly, and we can set up a JAX CI for using the oldest supported Cloud TPU (e.g. Cloud TPU at current time minus compatibility).

Future work

Experimental/under development provision

In PJRT C API

- An experimental version number will be added for experimental features.
- A separate struct will be added for experimental features.

Framework

- Experimental features can be changed with no forwards compatibility guarantees.
- Frameworks must maintain compatibility with features being missing until it comes out of experimental, at which point N week compatibility window begins.

Plugins

- Plugins do not need to implement experimental functions.
- Plugins need to set related fields to nullptr for unimplemented experimental features.

Users will have access to these experimental features

Optional features

As PJRT C APIs evolve, some APIs may be optional features. For a minor version update due to an added optional feature, the plugin still needs to make a new release, but does not need to implement C API of optional features. We may add a new struct for optional features, or leverage PJRT_Plugin_Attributes.

TF

The policies discussed above apply to TF. One main difference is that TF releases regularly with a quarterly cadence, and it may not use the new C APIs added even if it is released at a newer PJRT C API version. Plugins define its behavior when a plugin is newer than the framework.

PT/XLA

TBD.

References

- StableHLO versioning https://github.com/openxla/stablehlo/blob/main/docs/compatibility.md