Kotlin Remote Compilation

Michal Svec

Communication Overview (1)

```
service CompileService {
   rpc Compile (stream CompileRequestProto) returns (stream CompileResponseProto)
}
```

- streaming
  - bidirectional
  - connection is open during the entire time of compilation

Communication Overview (2)

CLIENT ⟹ **SERVER**

```
message CompilationMetadataProto{
  string project_name = 1;
  int32 total_files_to_send = 2;
  repeated string compiler_arguments = 3;
  oneof compilation_options {
    CompilationOptionsProto standard_compilation_options = 4;
    IncrementalCompilationOptionsProto incremental_compilation_options = 5;
  }
}
```

Communication Overview (3)

CLIENT ⇨ **SERVER**

```
message FileTransferRequestProto{
  string file_path = 1;
  string file_fingerprint = 2;
  repeated ArtifactTypeProto artifact_types = 3;
}
```

- fingerprint → sha256
- artifact types
  - file, dependency, compiler plugin, compilation result, classpath entry snapshot, shrunk classpath snapshot, IC cache
- kotlinx-coroutines, request are fired in parallel

Communication Overview (4)

CLIENT ⇦ **SERVER**

```
message FileTransferReplyProto{
  string file_path = 1;
  bool is_present = 2;
  repeated ArtifactTypeProto artifact_types = 3;
}
```

- `is_present = cache.contains(file fingerprint)`

Communication Overview (5)

CLIENT ⇨ **SERVER**

```
message FileChunkProto {
  string file_path = 1;
  repeated ArtifactTypeProto artifact_types = 2;
  bytes content = 3;
  bool is_directory = 4;
  bool is_last = 5;
}
```

- client sends files that are not available in the cache
- files can be large, they are sent in small chunks to prevent loading them to memory at once

Communication Overview (6)

CLIENT ⇦ **SERVER**

```
message CompilerMessageProto{
  CompilerMessageSeverityProto compiler_message_severity = 1;
  string message = 2;
  optional CompilerMessageSourceLocationProto compiler_message_source_location = 3;
}
```

- once all files are ready, compilation can be started

Communication Overview (7)

CLIENT ⇦ **SERVER**

```
message CompilationResultProto{
  uint32 exit_code = 1;
  CompilationResultSourceProto result_source = 2;
}
```

- result_source = cache or compiler

Communication Overview (8)

CLIENT ⇦ **SERVER**

```
message FileChunkProto {
  string file_path = 1;
  repeated ArtifactTypeProto artifact_types = 2;
  bytes content = 3;
  bool is_directory = 4;
  bool is_last = 5;
}
```

- we stream compiled files back to client
- in case of incremental compilation also IC cache

Incremental Compilation

- only modified files are sent server
  - we expect that rest of the files are available on the server from the previous run
  - in case they are not, server has an ability to request missing files
- client sends classpath snapshots to server
- server sends IC cache to client

```
message MissingArtifactsRequestProto{
  repeated ArtifactProto missing_artifacts = 1;
}
message ArtifactProto {
  string file_path = 1;
  repeated ArtifactTypeProto artifact_types = 2;
}
```

Cache

- /storage
  - /cache
    - /artifacts
      - /<file-fingerprint>
    - /tmp
      - /<random-filename>

- flat /artifacts folder + HashMap<Fingerprint, File>
- List<FileChunk> → file in /tmp → file in /artifacts
- source files, dependencies, compiler plugins
  - fingerprint = sha256 from file content
- compilation results
  - fingerprint = sha256 from all input files and compiler arguments

Compilation Workspace

- /storage
  - /workspace
    - /<user-id>
      - /<project-name>
        - /<client-absolute-filepath>

- all client paths in compiler arguments and compilation options need to be replaced with paths from compilation workspace

Kotlin Compilation

- we do not use existing Kotlin daemon
  - current Kotlin Daemon process uses RMI for interprocess communication

- in our case, everything runs in a single JVM process

Communication protocol

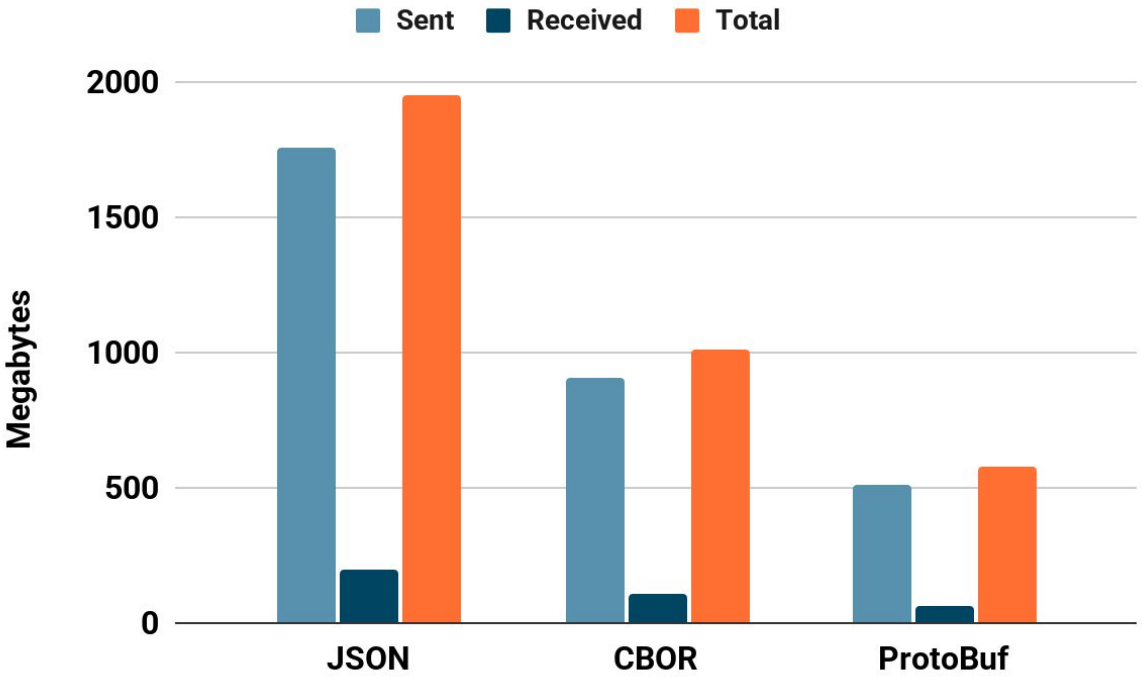
- all the logic stays the same, only the underlying transport differs

- **gRPC**
  - based on HTTP/2
  - ProtoBuf

- **kotlinx-rpc**
  - by default uses kRPC
    - based on WebSockets
  - allows easily swap serialization formats
    - JSON, CBOR, ProtoBuf
  - gRPC support is under development

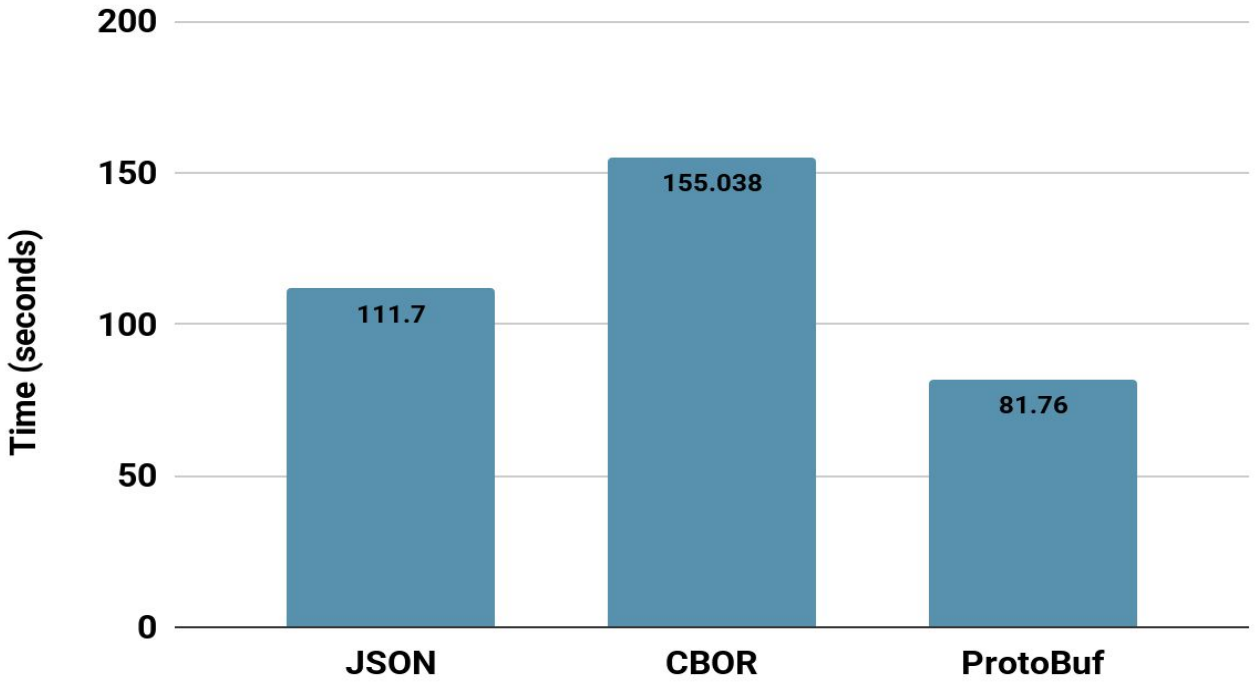
Serialization Formats Comparison

- all 202 Ktor K2JVM compilation tasks
- does not include any HTTP metadata, only size of data objects



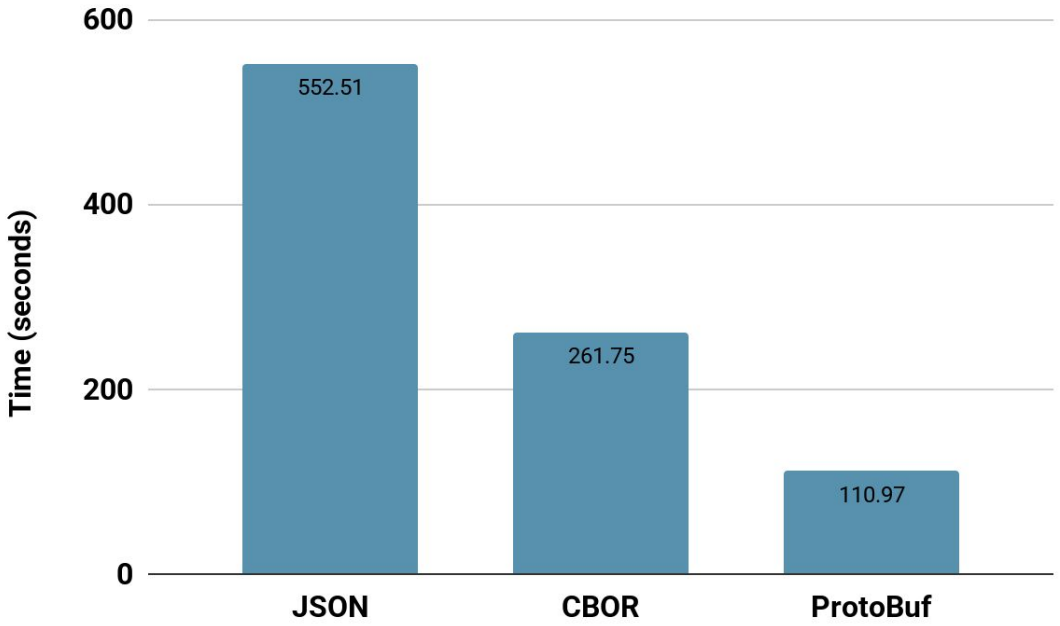
Serialization Formats Comparison (localhost)

- all Ktor K2JVM compilation tasks
- kotlinx-rpc, average of 15 iterations (1 warm up iteration before)



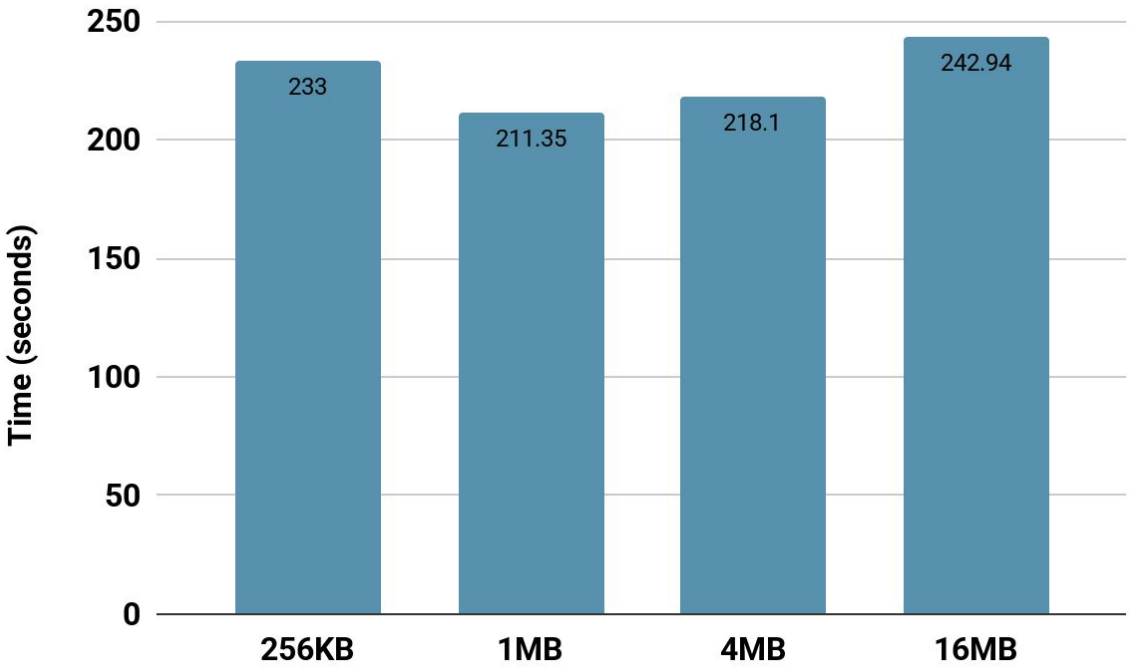
Serialization Formats Comparison (remote)

- **only first 50 Ktor K2JVM compilation tasks**
- kotlinx-rpc, average of 15 iterations (1 warm up iteration before)



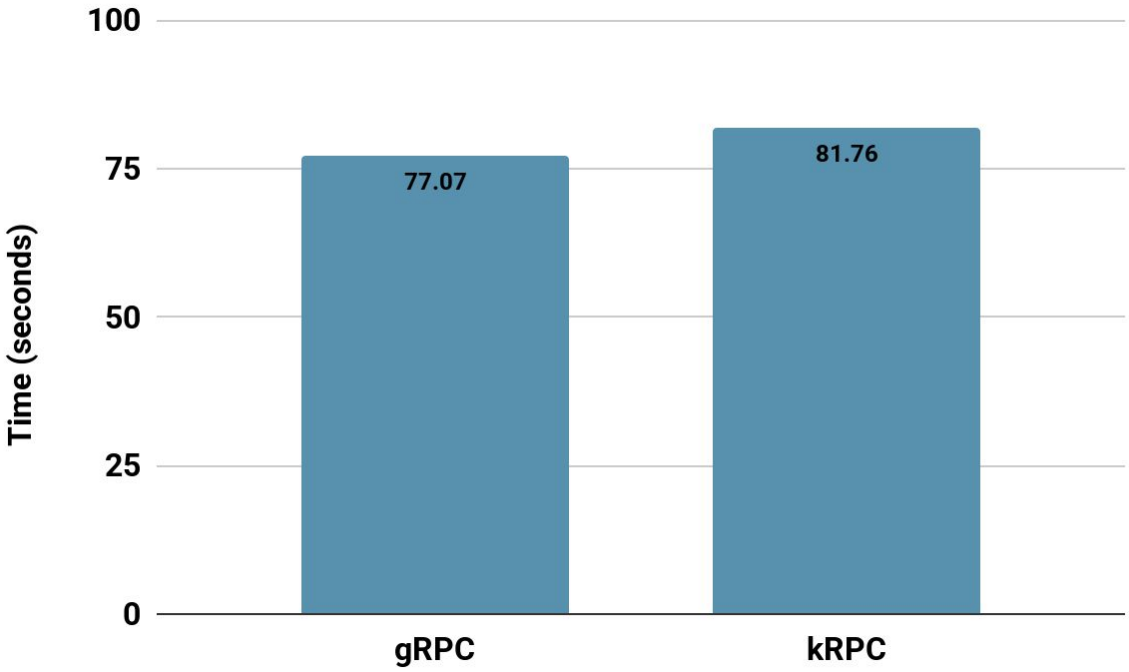
Chunk Size Comparison (remote)

- all Ktor K2JVM compilation tasks
- localhost, kotlinx-rpc, 15 iterations (1 warm up iteration before)



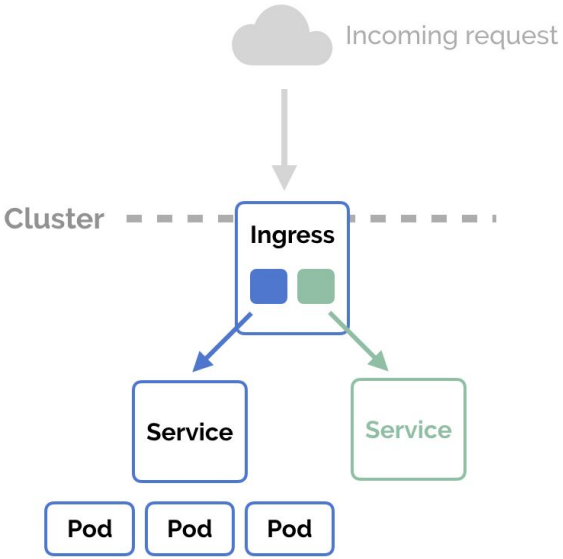
gRPC vs kotlinx-rpc (localhost)

- all Ktor K2JVM compilation tasks
- average of 15 iterations (1 warm up iteration before)



gRPC vs kotlinx-rpc (remote)

- all Ktor K2JVM tasks, average 15 iterations (1 warm up iteration before)
  - WebSockets **~4.20** minutes, gRPC **~8.10** minutes
- proper results are missing due to an issue with Nginx in Kubernetes cluster
- most likely Nginx configuration somehow messes up the HTTP/2



Limitations

- only JVM target is supported
- custom JDKs are not supported
  - we use only jvm-target flag to support older versions
- nonexistent integration to the existing infrastructure
  - difficult to trigger compilation, need to extract compilation task from Gradle debug log and feed that into compilation server
- only a single version of compiler plugins is supported
  - 1-1 correspondence between compiler and compiler plugins
  - we need to run multiple JVM processes with specific compiler version
    - then route compilation request to correct process
- …