Welcome to Avro!

Table of contents

1. Introduction

Avro is a data serialization system.

Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote procedure call (RPC).
- Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.

2. Schemas

Avro relies on *schemas*. When Avro data is read, the schema used when writing it is always present. This permits each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing.

When Avro data is stored in a file, its schema is stored with it, so that files may be processed later by any program. If the program reading the data expects a different schema this can be easily resolved, since both schemas are present.

When Avro is used in RPC, the client and server exchange schemas in the connection handshake. (This can be optimized so that, for most calls, no schemas are actually transmitted.) Since both client and server both have the other's full schema, correspondence between same named fields, missing fields, extra fields, etc. can all be easily resolved.

Avro schemas are defined with JSON . This facilitates implementation in languages that already have JSON libraries.

3. Comparison with other systems

Avro provides functionality similar to systems such as Thrift, Protocol Buffers, etc. Avro differs from these systems in the following fundamental aspects.

- *Dynamic typing*: Avro does not require that code be generated. Data is always accompanied by a schema that permits full processing of that data without code generation, static datatypes, etc. This facilitates construction of generic data-processing systems and languages.
- *Untagged data*: Since the schema is present when data is read, considerably less type

information need be encoded with data, resulting in smaller serialization size.
- *No manually-assigned field IDs*: When a schema changes, both the old and new schema are always present when processing data, so differences may be resolved symbolically, using field names.