UMD DATA605 - Big Data Systems

7.3: Serialization Formats

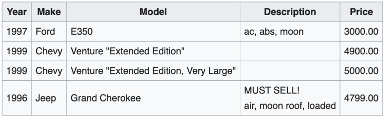
- **Instructor**: Dr. GP Saggese - gsaggese@umd.edu

Serialization Formats

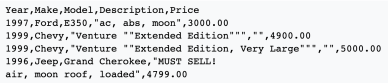
- Programs need to send data to each other (network, disk)
  - E.g., remote Procedure Calls (RPCs)
  - Recent technologies based on schemas
    - JSON, YAML, Protocol Buffer, Python Pickle
- Serialization formats are data models

Comma Separated Values (CSV)

- CSV stores data row-wise as text without schema
  - Each line is a data record
  - Records have fields separated by commas
- Pros
  - Very portable
    - Text format
    - Supported by all tools
  - Human-friendly
- Cons
  - Large footprint
    - Requires compression
  - Parsing is CPU intensive
  - No easy random access
  - Can't read only a subset of columns
  - No schema/types
    - Annotate CSV with schema
  - Mainly read-only, hard to modify



Year	Make	Model	Description	Price
1997	Ford	E350	ac, abs, moon	3000.00
1999	Chevy	Venture "Extended Edition"		4900.00
1999	Chevy	Venture "Extended Edition, Very Large"		5000.00
1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799.00



```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""","",4900.00
1999,Chevy,"Venture ""Extended Edition, Very Large""","",5000.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
```

(Apache) Parquet

- Parquet reads data tiles
- Supports multi-dimensional, nested data
  - Generalizes dataframes
- Column-storage
  - Stores each column together, uniform data type, compressed efficiently
- IO layer executes queries
  - Reads only necessary data chunks from disk
- **Pros**
  - 10x smaller than CSV
  - 10x faster with multi-threading
  - Read subset of columns and rows
- **Cons**
  - Binary, not human-friendly
  - Requires ingestion step to convert to Parquet
  - Mainly read-only, hard to modify

JSON

- JSON = JavaScript Object Notation
- Nested dictionaries and arrays
- Similar to XML
    - More human-readable
    - Less boilerplate
    - Executable in JavaScript and Python

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Protocol Buffers

- Developed by Google
- Open-source
- Represent data structures:
  - Language agnostic
  - Platform agnostic
  - Versioning
- Schema is mostly relational
  - Optional fields
  - Types
  - Default values
  - Structures
  - Arrays
- Schema specified using a `.proto` file
- Compiled by `protoc` to produce C++, Java, or Python code to initialize, read, serialize objects

```
import addressbook_pb2
person = addressbook_pb2.Person()
person.id = 1234
person.name = "John Doe"
person.email = "jdoe@example.com"
phone = person.phones.add()
phone.number = "555-4321"
phone.type = addressbook_pb2.Person.HOME
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }
  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2;
  }
  repeated PhoneNumber phones = 4;
}
```

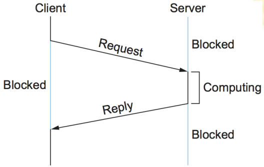
Serialization Formats

- Avro
  - Richer data structures
  - JSON-specified schema
- Thrift
  - Developed by Facebook
  - Now Apache project
  - More languages supported
  - Supports exceptions and sets

```
{
  "namespace": "example.avro",
  "type": "record",
  "name": "User",
  "fields": [
    {
      "name": "name",
      "type": "string"
    },
    {
      "name": "favorite_number",
      "type": [
        "int",
        "null"
      ]
    },
    {
      "name": "favorite_color",
      "type": [
        "string",
        "null"
      ]
    }
  ]
}
```

Remote Procedure Call

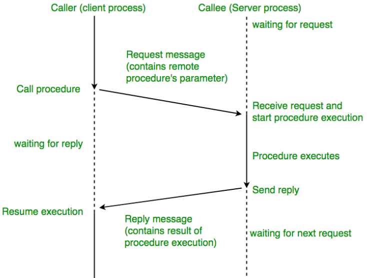
- **Remote Procedure Call** (RPC) requests services from programs on other computers, abstracting network communication
- **Goal**: Make remote calls like local procedure calls without network details
- **Problems**
  - Can't serialize pointers
  - Asynchronous communication
  - Failures and retry
- Used in distributed systems
  - E.g., microservices, cloud services, client-server applications
- Can be synchronous or asynchronous



SCIENCE ACADEMY

RPCs: Internals

- *Client procedure call*: Client calls stub function with arguments
- *Request marshalling*: Client stub serializes arguments for network transmission
- *Server communication*: Client's RPC runtime sends request to server
- *Server-side unmarshalling*: Server's RPC runtime deserializes arguments
- *Procedure execution*: Server calls procedure
- *Response marshalling*: Return values marshaled into response message
- *Client communication / response unmarshalling / return to client*: Return values passed back to client's stub call, execution



Remote procedure call model