Commit 6cb0131

Browse files

netfs authored and **tensorflow-copybara** committed on Aug 6, 2024 · ✓ 3 / 3

```
Improve handling of large JSON requests by:

 o Iteratively parse JSON.
 o Truncate large JSON strings to fixed size when composing error messages.

Thanks to Ori Hollander of the JFrog Security Research team for reporting.

PiperOrigin-RevId: 659735843
```

master · 2.19.0 ••• 2.18.0-rc0 1 parent ea02141 commit 6cb0131

Filter files...

- tensorflow_serving/util
  - BUILD
  - json_tensor.cc
  - json_tensor_test.cc

3 files changed +113 -23 lines changed

Search within code

tensorflow_serving/util/BUILD +1

```
@@ -340,6 +340,7 @@ cc_test(
340 340         "//tensorflow_serving/apis:regression_cc_proto",
341 341         "//tensorflow_serving/core/test_util:test_main",
342 342         "//tensorflow_serving/test_util",
    343 +       "@com_google_absl//absl/strings",
343 344         "@com_google_protobuf//:protobuf",
344 345         "@org_tensorflow//tensorflow/core:framework",
345 346         "@org_tensorflow//tensorflow/core:lib",
```

tensorflow_serving/util/json_tensor.cc +77 -23

```
@@ -16,6 +16,7 @@ limitations under the License.
16 16  #include "tensorflow_serving/util/json_tensor.h"
```

```
17  17
18  18    #include <algorithm>
    19  + #include <cstdint>
19  20    #include <cstdlib>
20  21    #include <functional>
21  22    #include <limits>
          @@ -26,7 +27,10 @@ limitations under the License.
26  27
27  28    #include "rapidjson/document.h"
28  29    #include "rapidjson/error/en.h"
    30  + #include "rapidjson/memorystream.h"
29  31    #include "rapidjson/prettywriter.h"
    32  + #include "rapidjson/rapidjson.h"
    33  + #include "rapidjson/reader.h"
30  34    #include "rapidjson/stringbuffer.h"
31  35    #include "absl/strings/escaping.h"
32  36    #include "absl/strings/match.h"
          @@ -165,20 +169,69 @@ bool WriteDecimal(RapidJsonWriter* writer, dtype val) {
165 169                              rapidjson::kNumberType);
166 170    }
167 171
    172 + // Write JSON string to a bounded string buffer of `max_bytes` size.
    173 + // NOTE: Output may contain truncated (and hence invalid) JSON.
    174 + //
    175 + // This class implements rapidjson::Handler concept, for use with
    176 + // rapidjson::Value::Accept() API.
    177 + class JsonWriterWithLimit {
    178 +  public:
    179 +   JsonWriterWithLimit(rapidjson::StringBuffer* buffer, RapidJsonWriter* writer,
    180 +                       int max_bytes)
    181 +       : buffer_(buffer), writer_(writer), max_bytes_(max_bytes) {};
    182 +
    183 +   bool Null() { return InLimit() ? writer_->Null() : false; }
    184 +   bool Bool(bool b) { return InLimit() ? writer_->Bool(b) : false; }
    185 +   bool Int(int i) { return InLimit() ? writer_->Int(i) : false; }
    186 +   bool Uint(unsigned u) { return InLimit() ? writer_->Uint(u) : false; }
    187 +   bool Int64(int64_t i64) { return InLimit() ? writer_->Int64(i64) : false; }
    188 +   bool Uint64(uint64_t u64) { return InLimit() ? writer_->Uint64(u64) : false; }
    189 +   bool Double(double d) { return InLimit() ? writer_->Double(d) : false; }
    190 +
    191 +   bool RawNumber(const rapidjson::MemoryStream::Ch* str,
    192 +                  rapidjson::SizeType length, bool copy = false) {
    193 +     return InLimit() ? writer_->RawNumber(str, length, copy) : false;
```

```
194 +   }
195 +
196 +   bool String(const rapidjson::MemoryStream::Ch* str,
197 +               rapidjson::SizeType length, bool copy = false) {
198 +     return InLimit() ? writer_->String(str, length, copy) : false;
199 +   }
200 +
201 +   bool StartObject() { return InLimit() ? writer_->StartObject() : false; }
202 +
203 +   bool Key(const rapidjson::MemoryStream::Ch* str, rapidjson::SizeType length,
204 +            bool copy = false) {
205 +     return InLimit() ? writer_->Key(str, length, copy) : false;
206 +   }
207 +
208 +   bool EndObject(rapidjson::SizeType memberCount = 0) {
209 +     return InLimit() ? writer_->EndObject(memberCount) : false;
210 +   }
211 +
212 +   bool StartArray() { return InLimit() ? writer_->StartArray() : false; }
213 +
214 +   bool EndArray(rapidjson::SizeType memberCount = 0) {
215 +     return InLimit() ? writer_->EndArray(memberCount) : false;
216 +   }
217 +
218 +  private:
219 +   bool InLimit() { return buffer_->GetSize() < max_bytes_; }
220 +   const rapidjson::StringBuffer* const buffer_;
221 +   RapidJsonWriter* const writer_;
222 +   const int max_bytes_;
223 + };
224 +
225 + constexpr int kMaxJsonDebugStringBytes = 256;
226 +
168 227   // Stringify JSON value (only for use in error reporting or debugging).
169     - string JsonValueToString(const rapidjson::Value& val) {
170     -   // TODO(b/67042542): Truncate large values.
228 + // Large JSON objects are truncated to kMaxJsonDebugStringBytes.
229 + string JsonValueToDebugString(const rapidjson::Value& val) {
171 230     rapidjson::StringBuffer buffer;
172 231     RapidJsonWriter writer(buffer);
173     -   // Write decimal numbers explicitly so inf/nan's get printed
174     -   // correctly. RapidJsonWriter() does not write these correctly.
175     -   if (val.IsFloat()) {
176     -     WriteDecimal(&writer, val.GetFloat());
```

```
177        -   } else if (val.IsDouble()) {
178        -     WriteDecimal(&writer, val.GetDouble());
179        -   } else {
180        -     val.Accept(writer);
181        -   }
       232 +   writer.SetFormatOptions(rapidjson::kFormatSingleLineArray);
       233 +   JsonWriterWithLimit j(&buffer, &writer, kMaxJsonDebugStringBytes);
       234 +   val.Accept(j);
182    235      return buffer.GetString();
183    236    }
184    237
       @@ -208,24 +261,25 @@ bool IsShapeEqual(const TensorShapeProto& lhs, const
       TensorShapeProto& rhs) {
208    261
209    262    Status TypeError(const rapidjson::Value& val, DataType dtype) {
210    263      return errors::InvalidArgument(
211        -         "JSON Value: ", JsonValueToString(val), " Type: ", JsonTypeString(val),
       264 +         "JSON Value: ", JsonValueToDebugString(val),
       265 +         " Type: ", JsonTypeString(val),
212    266           " is not of expected type: ", DataTypeString(dtype));
213    267    }
214    268
215    269    Status Base64FormatError(const rapidjson::Value& val) {
216        -   return errors::InvalidArgument("JSON Value: ", JsonValueToString(val),
       270 +   return errors::InvalidArgument("JSON Value: ", JsonValueToDebugString(val),
217    271                                     " not formatted correctly for base64 data");
218    272    }
219    273
220    274    template <typename... Args>
221    275    Status FormatError(const rapidjson::Value& val, Args&&... args) {
222        -   return errors::InvalidArgument("JSON Value: ",
223        -     JsonValueToString(val), " ", std::forward<Args>(args)...);
       276 +   return errors::InvalidArgument("JSON Value: ", JsonValueToDebugString(val),
       277 +                                  " ", std::forward<Args>(args)...);
224    278    }
225    279
226    280    Status FormatSignatureError(const rapidjson::Value& val) {
227    281      return errors::InvalidArgument(
228        -         "JSON Value: ", JsonValueToString(val),
       282 +         "JSON Value: ", JsonValueToDebugString(val),
229    283           " not formatted correctly. 'signature_name' key must be a string value.");
230    284    }
231    285
```

```
@@ -281,7 +335,7 @@ Status AddValueToTensor(const rapidjson::Value& val, DataType dtype,
281 335
282 336       default:
283 337         return errors::Unimplemented(
284     -           "Conversion of JSON Value: ", JsonValueToString(val),
    338 +           "Conversion of JSON Value: ", JsonValueToDebugString(val),
285 339             " to type: ", DataTypeString(dtype));
286 340       }
287 341       return OkStatus();
@@ -343,7 +397,7 @@ Status FillTensorProto(const rapidjson::Value& val, int level,
DataType dtype,
343 397       // at same (leaf) level equal to the rank of the tensor.
344 398       if (level != rank) {
345 399         return errors::InvalidArgument(
346     -           "JSON Value: ", JsonValueToString(val),
    400 +           "JSON Value: ", JsonValueToDebugString(val),
347 401             " found at incorrect level: ", level + 1,
348 402             " in the JSON DOM. Expected at level: ", rank);
349 403       }
@@ -426,10 +480,10 @@ Status ParseJson(const absl::string_view json, rapidjson::Document*
doc) {
426 480     rapidjson::MemoryStream ms(json.data(), json.size());
427 481     rapidjson::EncodedInputStream<rapidjson::UTF8<>, rapidjson::MemoryStream>
428 482         jsonstream(ms);
429     -   // TODO(b/67042542): Switch to using custom stack for parsing to protect
430     -   // against deep nested structures causing excessive recursion/SO.
431     -   if (doc->ParseStream<rapidjson::kParseNanAndInfFlag>(jsonstream)
432     -           .HasParseError()) {
    483 +   constexpr auto parse_flags = rapidjson::kParseIterativeFlag |
    484 +                                rapidjson::kParseNanAndInfFlag |
    485 +                                rapidjson::kParseStopWhenDoneFlag;
    486 +   if (doc->ParseStream<parse_flags>(jsonstream).HasParseError()) {
433 487       return errors::InvalidArgument(
434 488           "JSON Parse error: ", rapidjson::GetParseError_En(doc->GetParseError()),
435 489           " at offset: ", doc->GetErrorOffset());
@@ -514,7 +568,7 @@ Status FillTensorMapFromInstancesList(
514 568         if (input_names != object_keys) {
515 569           return errors::InvalidArgument(
516 570               "Failed to process element: ", tensor_count,
517     -             " of 'instances' list. JSON object: ", JsonValueToString(elem),
    571 +             " of 'instances' list. JSON object: ", JsonValueToDebugString(elem),
518 572               " keys must be equal to: ", absl::StrJoin(input_names, ","));
519 573         }
```

```
520  574        } else {
```

tensorflow_serving/util/json_tensor_test.cc +35

```
@@ -16,6 +16,7 @@ limitations under the License.
16  16   #include "tensorflow_serving/util/json_tensor.h"
17  17
18  18   #include <functional>
    19 + #include <string>
19  20
20  21   #include "google/protobuf/message.h"
21  22   #include "google/protobuf/text_format.h"

@@ -83,6 +84,40 @@ TEST(JsontensorTest, SingleUnnamedTensor) {
83  84       )"));
84  85   }
85  86
    87 + TEST(JsontensorTest, DeeplyNestedWellFormed) {
    88 +   TensorInfoMap infomap;
    89 +   ASSERT_TRUE(
    90 +       TextFormat::ParseFromString("dtype: DT_INT32", &infomap["default"]));
    91 +
    92 +   PredictRequest req;
    93 +   JsonPredictRequestFormat format;
    94 +   std::string json_req = R"({"instances":[1], "nested":)";
    95 +   json_req.append(500000, '[');
    96 +   json_req.append(500000, ']');
    97 +   json_req.append("}");
    98 +   TF_EXPECT_OK(
    99 +       FillPredictRequestFromJson(json_req, getmap(infomap), &req, &format));
   100 +   auto tmap = req.inputs();
   101 +   EXPECT_EQ(tmap.size(), 1);
   102 + }
   103 +
   104 + TEST(JsontensorTest, DeeplyNestedMalformed) {
   105 +   TensorInfoMap infomap;
   106 +   ASSERT_TRUE(
   107 +       TextFormat::ParseFromString("dtype: DT_INT32", &infomap["default"]));
   108 +
   109 +   PredictRequest req;
   110 +   JsonPredictRequestFormat format;
   111 +   std::string json_req = R"({"signature_name":)";
   112 +   json_req.append(500000, '[');
```

```
113 +   json_req.append(500000, ']');
114 +   json_req.append("}");
115 +   auto status =
116 +       FillPredictRequestFromJson(json_req, getmap(infomap), &req, &format);
117 +   ASSERT_TRUE(errors::IsInvalidArgument(status));
118 +   EXPECT_THAT(status.message(), HasSubstr("key must be a string value"));
119 + }
120 +
86  121   TEST(JsontensorTest, MixedInputForFloatTensor) {
87  122     TensorInfoMap infomap;
88  123     ASSERT_TRUE(
```

Comments 0

Please sign in to comment.