Evaluation Workload

Ralf Diestelkämper

April 1, 2019

This document describes the workload applied to evaluate Pebble. The workload consists of ten scenarios. Each scenario consists of a short description, a Spark program, and tree-pattern definition. The first five scenarios (T1 - T5) run on a Twitter dataset obtained via the public Twitter API (`https://developer.twitter.com/en/docs.html`). The other five scenarios (D1-D5) run on the DBLP dataset obtained from `https://dblp.org/xml/`.

1 Twitter evaluation scenarios

Scenario T1

The program unnests the mentioned users and filters tweets containing the term "good". Then it groups the tweets by the mentioned users and collects a bag of complex types "origin" holding the tweeted texts, the tweeting user, and tweet entities as children. When we inspect the program's result, we identify that a child of the nested entities attribute holds an attribute hashtags, which appears to be null in many result items. Employing structural provenance, we investigate, whether non-null hashtags attributes exist in result items.

```
val twitter = loadTweets()
var mentioned = twitter.withColumn("mentioned_user", explode($"entities.user_mentions"))
val good_tweets = mentioned.filter($"text".contains("good"))
val good_tweets_selected = good_tweets.select($"mentioned_user.id".alias("mentioned_user"),
                  struct($"user", $"text", $"entities").alias("origin"))
var good_inversed = good_tweets_selected.groupBy($"mentioned_user").agg(collect_list($"origin").alias("origin"))
```

```
var twig = new Twig()
val root = twig.createNode("root")
val origin = twig.createNode("origin")
val element = twig.createNode("element")
val hashtags = twig.createNode("hashtags", 1, 1000)

twig = twig.createEdge(root, origin, false)
twig = twig.createEdge(origin, element, false)
twig = twig.createEdge(element, hashtags, true)
```

Scenario T2

This program represents important steps of ETL processes for relational databases. It flattens the nested lists "hashtags", "media", and "user_mentions". On the flattened result, we like to find and trace back all items associated with the user name "cool", regardless whether they are mentioned or authoring users.

```
val twitter = loadTweets()
val twitter_truncated = twitter.select($"created_at", $"entities", $"quoted_status_id", $"favorite_count",
    $"favorited", $"id", $"lang", $"possibly_sensitive", $"source", $"text", $"timestamp_ms", $"truncated", $"user")
val flattened_ht = twitter_truncated.withColumn("hashtags", explode($"entities.hashtags"))
val flattened_ht_media = flattened_ht.withColumn("media", explode($"entities.media"))
val flattened_ht_media_url = flattened_ht_media.withColumn("urls", explode($"entities.urls"))
val flattened_ht_media_url_user = flattened_ht_media_url.withColumn("user_mentions", explode($"entities.user_mentions"))
```

```
var twig = new Twig()
val root = twig.createNode("root")
val name = twig.createNode("name", condition = "cool")
twig = twig.createEdge(root, name, true)
```

Scenario T3

The program associates users with tweets they have been tweeting or have been mentioned in. The inspection of the result reveals that some texts occur multiple times. In the result we identify a user "Carlos-123" holding two tweets with the same texts. Structural provenance reveals that the user has mentioned multiple other users. However, the latter two options would have been part of an answers based solely on prospective provenance.

```
val twitter = loadTweets()
var mentioned = twitter.withColumn("mentioned_user", explode($"entities.user_mentions"))
val extracted_mentioned_users = mentioned.select($"created_at", $"text", $"id".alias("tid"),
  $"mentioned_user.id", $"mentioned_user.id_str", $"mentioned_user.name", $"mentioned_user.screen_name")
val extracted_users = twitter.select($"created_at", $"text", $"id".alias("tid"),
                $"user.id", $"user.id_str", $"user.name", $"user.screen_name")
val all_users = extracted_users.union(extracted_mentioned_users)
val restructured_users = all_users.select(struct($"id", $"id_str", $"name", $"screen_name").alias("user"),
  struct($"created_at", $"text", $"tid").alias("tweet"))
val res = restructured_users.groupBy($"user").agg(collect_list($"tweet").alias("tweets"))
```

```
val root = twig.createNode("root")
val name = twig.createNode("name", condition = "Carlos-123")
val text = twig.createNode("text", 2, 100)
twig = twig.createEdge(root, name, true)
twig = twig.createEdge(root, text, true)
```

Scenario T4

The program aims at associating all occurring hashtags with the authoring and mentioned users. In the result, the authoring users are direct children of each result item, whereas mentioned users reside nested "entities" list holding "user_mentions" lists. The provenance query addresses this structurally unexpected result and reveals that the mentioned users reside in nested lists in the input data. The program takes these lists as they are and nest them in a new collection instead of wrapping all mentioned users in one list as initially intended.

```
val twitter = loadTweets()
val flattened_ht = twitter.withColumn("hashtags", explode($"entities.hashtags"))
val ht2 = flattened_ht.groupBy($"hashtags").agg(
                collect_list($"user").alias("users"), collect_list($"entities").alias("entitiesList"))
```

```
val entitiesList = twig.createNode("entitiesList")
val element1 = twig.createNode("element", 1, 100)
val user_mentions = twig.createNode("user_mentions")
val element2 = twig.createNode("element", 1, 100)

twig = twig.createEdge(root, entitiesList, false)
twig = twig.createEdge(entitiesList, element1, false)
twig = twig.createEdge(element1, user_mentions, false)
twig = twig.createEdge(user_mentions, element2, false)
```

Scenario T5

The final twitter scenario finds all users that did not only tweet about "BTS", but also got mentioned in such a tweet. For that purpose, the program unnests the "mentioned_users" and joins them with the users tweeting. Unexpectly, the result contains tweet texts not containing "BTS". Provenance reveals that the text originates in the tweets of the (unfiltered) authoring users, not the mentioned users.

```
val twitter = loadTweets()
var mentioned = twitter.withColumn("mentioned_user", explode($"entities.user_mentions"))
mentioned = mentioned.filter($"text".contains("BTS"))
val small_mention = mentioned.select($"mentioned_user", $"text".alias("mentioned_text"), $"id".alias("mentioned_id"))
val joined = small_mention.join(twitter, $"mentioned_user.id" === $"user.id")
```

```
val root = twig.createNode("root")
val text = twig.createNode("text", condition = "notcontainsBTS")
val name = twig.createNode("screen_name",  condition="BTS_twt" )

twig = twig.createEdge(root, text, false)
twig = twig.createEdge(root, name, true)
```

2 DBLP evaluation scenarios

Scenario D1

The program associates all inproceedings from 2015 with the their according proceeding(s). The result contains an item with name "John Miller" having a year of 2016. The structural provenance reveals that the year originates in the proceedings and is independent of the inproceedings year.

```
val proceedings = loadProceedings()
val inproceedings = loadInproceedings()
var inproceedings_processed = inproceedings.withColumn("crf", explode($"crossref"))
inproceedings_processed = inproceedings_processed.filter($"year" === 2015)
inproceedings_processed = inproceedings_processed.select($"author".alias("ip_author"), $"title".alias("ip_title"), $"crf")
val inproceedings_proceedings = proceedings.join(inproceedings_processed,
                proceedings("_key") === inproceedings_processed("crf"))
```

```
val root = twig.createNode("root")
val year = twig.createNode("year", condition = "2016")
val author = twig.createNode("ip_author")
val element = twig.createNode("element")
val value = twig.createNode("_VALUE", 1, 100, condition = "John Miller")

twig = twig.createEdge(root, year, false)
twig = twig.createEdge(root, author, true)
twig = twig.createEdge(author, element, false)
twig = twig.createEdge(element, value, false)
```

Scenario D2

This program computes the union of all conference proceedings and articles. It returns not only their title and authors, but also the series title (i.e the journal name, or conference name). We trace all result items back to the input that contain "ACM" in the series title and have 10 or more authors.

```
val proceedings = loadProceedings()
val inproceedings = loadInproceedings()
val article = loadArticle()
var inproceedings_processed = inproceedings.withColumn("crf", explode($"crossref"))
inproceedings_processed = inproceedings_processed.select($"author", $"title", $"crf")
var proceedings_processed = proceedings.select($"_key", $"title".alias("series_title"))
var all_proceedings = inproceedings_processed.join(proceedings_processed,
  proceedings_processed("_key") === inproceedings_processed("crf"))
all_proceedings = all_proceedings.select($"author", $"title", $"series_title")
var article_processed = article.select($"author", $"title", $"journal".alias("series_title"))
var all = all_proceedings.union(article_processed)
```

```
val root = twig.createNode("root")
val author = twig.createNode("author")
val element = twig.createNode("element", 10, 10000)
val series_title = twig.createNode("series_title", condition="containsACM")
twig = twig.createEdge(root, author, false)
twig = twig.createEdge(author, element, false)
twig = twig.createEdge(root, series_title, false)
```

Scenario D3

This scenario computes a list of aliase, co-authors, and works per author. The tree-pattern tracks the name "Tommaso_0413" that occurs in the aliase and co-authors list.

```
var inproceedings = loadInproceedings()
inproceedings = inproceedings.withColumn("i_author", explode($"author"))
inproceedings = inproceedings.select($"i_author.name".alias("name"), $"key".alias("ikey"), $"title", $"i_author", $"author")
var authors = Provenance.readJson(authorPath, spark)
authors = authors.withColumn("u_author", explode($"author"))
authors = authors.select($"u_author.name".alias("name"), $"key".alias("akey"), $"u_author")
var joined = authors.join(inproceedings_selected, Seq("name"))
joined = joined.withColumn("co_author", explode($"author"))
var res = joined.groupBy("akey").agg(collect_set("u_author").alias("alias"),
                collect_set("co_author").alias("co_authors"), collect_set("title").alias("works"))
```

```
var twig = new Twig()
val root = twig.createNode("root")
val alias = twig.createNode("alias")
val element1 = twig.createNode("element")
val value1 = twig.createNode("_VALUE", 1, 100, condition = "Tommaso_0413")
val co_authors = twig.createNode("co_authors")
val element2 = twig.createNode("element")
val value2 = twig.createNode("_VALUE", 1, 100, condition = "Tommaso_0413")

twig = twig.createEdge(root, alias, false)
twig = twig.createEdge(alias, element1, false)
twig = twig.createEdge(element1, value1, false)
twig = twig.createEdge(root, co_authors, false)
twig = twig.createEdge(co_authors, element2, false)
twig = twig.createEdge(element2, value2, false)
```

Scenario D4

The program collects for each proceeding a nested list of all associated inproceedings. With the help of the provenance question, we track all "high quality" result items which hold values in important attributes like publisher, editor and series. In addition, each proceeding must have more than 50 inproceedings.

```
val proceedings = loadProceedings()
val inproceedings = loadInproceedings()
var inproceedings_flattened = inproceedings.withColumn("crf", explode($"crossref"))
inproceedings_flattened = inproceedings_flattened.select($"crf".alias("_key"),
                struct($"_key", $"author", $"title", $"_mdate").alias("inproc"))
val inproceedings_grouped = inproceedings_flattened.groupBy($"_key")
val inproceedings_restructured = inproceedings_grouped.agg(collect_list($"inproc").alias("inproc"))
val proceedings_with_inproceedings = proceedings.join(inproceedings_restructured, Seq("_key"))
```

```
val root = twig.createNode("root")
val author = twig.createNode("author")
val publisher = twig.createNode("publisher")
val editor = twig.createNode("editor")
val series = twig.createNode("series")
val inproc = twig.createNode("inproc")
val element = twig.createNode("element", 50, 10000)
twig = twig.createEdge(root, author, true)
twig = twig.createEdge(root, editor, true)
twig = twig.createEdge(root, publisher, true)
twig = twig.createEdge(root, series, true)
twig = twig.createEdge(root, inproc, false)
twig = twig.createEdge(inproc, element, false )
```

Scenario D5

The last DBLP scenario extends the previous scenario. Given the proceedings and nested inproceedings, we apply a map operator with a user defined function, which returns the number of authors per proceeding. We track all items back to the input, which have "vldb" in their name and more than 50 (nondistinct) authors.

```
val inproceedings = loadInproceedings()
var inproceedings_flattened = inproceedings.withColumn("crf", explode($"crossref"))
inproceedings_flattened = inproceedings_flattened.select($"crf".alias("_key"),
                struct($"_key", $"author", $"title", $"_mdate").alias("inproc"))
val inproceedings_grouped = inproceedings_flattened.groupBy($"_key")
val inproceedings_restructured = inproceedings_grouped.agg(collect_list($"inproc").alias("inproc"))
val proceedings_with_inproceedings = proceedings.join(inproceedings_restructured, Seq("_key"))
val authors_per_proceedigns = proceedings_with_inproceedings.map(row => {
  var cnt = 0; val iidx = row.fieldIndex("inproc"); val kidx = row.fieldIndex("_key")
  for(inproc <- row.getSeq[Row](iidx)) {
    if (inproc != null) {
      val aidx = inproc.fieldIndex("author")
      val author = inproc.getSeq[Row](aidx)
      if (author != null) { cnt += author.size } } }
  Tuple2(row.getString(kidx), cnt) })
```

```
val root = twig.createNode("root")
val e1 = twig.createNode("_1", condition = "containsvldb")
val e2 = twig.createNode("_2", condition = "gtgtgtgt200")
twig = twig.createEdge(root, e1, false)
twig = twig.createEdge(root, e2, false)
```