# Database - Bug #6887

## improve performance of dataset/table[-handle] parameters

10/26/2022 10:48 AM - Constantin Asofiei

| | | | |
|---|---|---|---|
| **Status:** | Review | **Start date:** | |
| **Priority:** | Normal | **Due date:** | |
| **Assignee:** | | **% Done:** | 0% |
| **Category:** | | **Estimated time:** | 0.00 hour |
| **Target version:** | | | |
| **billable:** | No | **case_num:** | |
| **vendor_id:** | GCD | **version:** | |

**Description**

### History

#### #1 - 10/26/2022 10:55 AM - Constantin Asofiei

In a large application, there is this JMX reported times (count/millis/nanos):

```
OrmDataSetParam/9056/177.895895/177895895
OrmDynamicDataSetParam/1783/202.068386/202068386
OutputDataSetHandleCopier/206/144.215879/144215879
```

Most of the time is spent in StaticDataSet.bind, copyDataSetImpl and TemporaryBuffer.copyAllRows.

#### #2 - 11/13/2022 11:14 PM - Eric Faulhaber

*- Assignee set to Boris Schegolev*

Constantin, is this something you think can reasonably be reproduced with some simplified test cases, which would illustrate the issue with the 3 methods you noted, or do you think it is better for Boris to set up the large application you were using?

#### #3 - 11/14/2022 12:04 PM - Constantin Asofiei

A starting point for a test to check the performance is this, made of two programs:

1. dstest.p which defines internal procedures, accepting dataset-handle and a dataset parameter:

```
def temp-table tt1 field f1 as int.
def dataset ds1 for tt1.

procedure proc0a.
   def input-output parameter dataset-handle dsh1.
   delete object dsh1 no-error.
end.
procedure proc0b.
   def output  parameter dataset-handle dsh1.
```

```
   delete object dsh1 no-error.
end.
procedure proc0c.
   def input parameter dataset-handle dsh1.
   delete object dsh1 no-error.
end.

procedure proc1.
    def input parameter dataset for ds1.
end.
```

2. a dsrun.p program, which is being executed:

```
def temp-table tt1 field f1 as int.
def dataset ds1 for tt1.

def var i as int.
def var l1 as int64.
def var l2 as int64.
def var hds as handle.
def var h as handle.

hds = dataset ds1:handle.
run dstest.p persistent set h.

do i = 1 to 10:
  create tt1.
  tt1.f1 = i.
  release tt1.
end.

// run once to warm-up the FWD server
run proc0a in h (input-output dataset-handle hds).
run proc0b in h (output dataset-handle hds).
run proc0c in h (input dataset-handle hds).
run proc1 in h (input dataset ds1 by-reference).

l1 = etime.
do i = 1 to 2000:
  run proc0a in h (input-output dataset-handle hds).
end.
l2 = etime.
message "input-output dataset-handle" l2 - l1 "ms".

l1 = etime.
do i = 1 to 2000:
  run proc0b in h (output dataset-handle hds).
end.
l2 = etime.
message "output dataset-handle" l2 - l1 "ms".

l1 = etime.
do i = 1 to 2000:
  run proc0c in h (input dataset-handle hds).
end.
l2 = etime.
message "input dataset-handle" l2 - l1 "ms".

l1 = etime.
do i = 1 to 10000:
  run proc1 in h (input dataset ds1 by-reference).
end.
l2 = etime.

message "input by-reference dataset" l2 - l1 "ms".
```

The tt1 temp-table needs to be expanded to hold 50-100 fields (like f1, f2, f3, ..., f50).

proc1 is being called with BY-REFERENCE and this can be tested to check the 'bind' issue mentioned in first note.  The other procedures are related to copying data, creating dynamic dataset, etc.

Profiling can be used when running the FWD server with dsrun.p, to see the hot-spots in the FWD runtime.

#### #4 - 11/14/2022 12:28 PM - Constantin Asofiei

I've edited previous note to add delete object hds1. to dstest.p - with this, OE is in the 200ms range, while FWD is in the 4s range for INPUT-OUTPUT DATASET-HANDLE.

#### #5 - 02/16/2023 03:23 PM - Eric Faulhaber

Has any of this implementation been touched by the recent waves of performance work in 3821c and 6129c, or are we essentially in the same place with this task as we were 3 months ago? Has any of the recent work in #6444 impacted this task?

#### #6 - 02/16/2023 04:31 PM - Boris Schegolev

I didn't make any changes for this task. Not sure if there are any relevant changes coming from #6444.

#### #7 - 02/16/2023 05:36 PM - Ovidiu Maxiniuc

The methods/attributes implemented by #6444 are not related to parameters. They perform the same regardless the dataset/temp-table object was received or is a local instance.

#### #8 - 03/13/2023 06:47 PM - Boris Schegolev

*- Status changed from New to WIP*

I did more testing on provided code. The INPUT-OUTPUT DATASET-HANDLE is slightly slower in FWD, but it's definitely not in whole seconds any more. I will try debugging/profiling just to see if I can find something suspicious, but generally the current version must have improved a lot.

#### #9 - 03/15/2023 01:18 PM - Boris Schegolev

I found a few points where we spend most CPU time:

1. Reflexive invocation java.lang.reflect.Method.invoke() - this is out of our control.
2. BlockManager.processBody() - this is a complex method, may be a candidate for optimization.
3. DataSet.createDynamicDataSetImpl() and DataSet.copyDatasetImpl() - these do a lot of heavy lifting. I can make minor changes here right away, not sure about results.

The most important thing is that I don't want to break any logic that's already implemented. If you have any inputs on these methods (i.e. what can be optimized) you are welcome to share :)

#### #10 - 03/16/2023 02:31 AM - Eric Faulhaber

Constantin Asofiei wrote:

> In a large application, there is this JMX reported times (count/millis/nanos):
>
> ```
> OrmDataSetParam/9056/177.895895/177895895
> OrmDynamicDataSetParam/1783/202.068386/202068386
> OutputDataSetHandleCopier/206/144.215879/144215879
> ```
>
> Most of the time is spent in StaticDataSet.bind, copyDataSetImpl and TemporaryBuffer.copyAllRows.

Constantin, can you gather these same statistics for a more recent run of the same tests, under the same conditions, through that large application, using a version of FWD (presumably trunk) with the last few months of optimizations? I don't want Boris spinning his wheels, if we've already flattened the curve here.

#### #11 - 03/16/2023 02:59 AM - Eric Faulhaber

Boris Schegolev wrote:

> I found a few points where we spend most CPU time:
>
> 1. Reflexive invocation java.lang.reflect.Method.invoke() - this is out of our control.

Not entirely. We've been trying different approaches here. See #6819, #6939, for example. We control the proxy implementation and the manner in which these methods are invoked.

> 2. BlockManager.processBody() - this is a complex method, may be a candidate for optimization.

This is the main callback into converted business logic in a block body, so we have to differentiate between time spent in converted code and the overhead added by this method itself. You can see some of the tasks around this area in #4061. If you see some noteworthy overhead in the "self" portion of this method that we haven't called out in one of these tasks, please document it.

> 3. DataSet.createDynamicDataSetImpl() and DataSet.copyDatasetImpl() - these do a lot of heavy lifting. I can make minor changes here right away, not sure about results.
>
> The most important thing is that I don't want to break any logic that's already implemented. If you have any inputs on these methods (i.e. what can be optimized) you are welcome to share :)

It depends what appears as a bottleneck. If there's something implemented inefficiently there that pops out in your profiling, we should discuss it. I definitely agree that it is critical not to regress this functionality; testing will determine that.

It sounds like you're not finding obvious bottlenecks at this point, using the latest FWD code. Let's see what Constantin comes back with on the JMX results before you invest more time in this task.

#### #12 - 03/16/2023 11:53 AM - Constantin Asofiei

This is the entire JMX result for the large application:

```
[2023-03-16 17:48:10] FWD:ServerStateS11n (count/ms/ns):     7789/0/593733
[2023-03-16 17:48:10] FWD:MessagePayloadS11n (count/ms/ns):     15778/51/51494921
[2023-03-16 17:48:10] FWD:objToByteArray (count/ms/ns):     7889/23/23390785
[2023-03-16 17:48:10] FWD:sslSend (count/ms/ns):     7789/73/73214795
[2023-03-16 17:48:10] FWD:sslWrap (count/ms/ns):     7789/69/69436147
[2023-03-16 17:48:10] FWD:sslUnwrap (count/ms/ns):     7825/51/51494825
[2023-03-16 17:48:10] FWD:ContextLocalGet (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_SymRes (count/ms/ns):     48/0/336758
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_SchemaDict (count/ms/ns):     24/0/235848
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Parse (count/ms/ns):     24/0/932225
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Parse2 (count/ms/ns):     24/3/3719556
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Setup (count/ms/ns):     24/0/975427
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Annotation (count/ms/ns):     1/2/2394631
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Base (count/ms/ns):     1/0/231459
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Core (count/ms/ns):     1/1/1833734
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Post (count/ms/ns):     1/0/146279
[2023-03-16 17:48:10] FWD:OrmDynQueryProcess_Intern (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:OrmDynParseFind (count/ms/ns):     708/15/15914381
[2023-03-16 17:48:10] FWD:OrmDynParseQuery (count/ms/ns):     5402/67/67023819
[2023-03-16 17:48:10] FWD:OrmHydrateRecord (count/ms/ns):     52562/450/450688510
[2023-03-16 17:48:10] FWD:OrmFqlParse (count/ms/ns):     1604/90/90533902
[2023-03-16 17:48:10] FWD:OrmTempTableQuery (count/ms/ns):     110231/387/387536626
[2023-03-16 17:48:10] FWD:OrmDynQueryInterpret (count/ms/ns):     6865/93/93407412
[2023-03-16 17:48:10] FWD:OrmHqlHelperParse (count/ms/ns):     37/1/1359048
[2023-03-16 17:48:10] FWD:OrmPreselectQueryAssemble (count/ms/ns):     18340/187/187624329
[2023-03-16 17:48:10] FWD:OrmCreateDynamicTable (count/ms/ns):     1912/11/11026130
[2023-03-16 17:48:10] FWD:OrmCreateDynamicBuffer (count/ms/ns):     2071/22/22349270
[2023-03-16 17:48:10] FWD:OrmBufferDefineTemp (count/ms/ns):     46093/297/297156575
[2023-03-16 17:48:10] FWD:OrmBufferDefinePerm (count/ms/ns):     12443/39/39851207
[2023-03-16 17:48:10] FWD:OrmBufferDelete (count/ms/ns):     60426/380/380284906
[2023-03-16 17:48:10] FWD:OrmTableMapperMap (count/ms/ns):     81573/26/26045246
[2023-03-16 17:48:10] FWD:OrmTempTableParam (count/ms/ns):     11874/68/68859566
[2023-03-16 17:48:10] FWD:OrmDynamicTableParam (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:OrmDataSetParam (count/ms/ns):     9135/113/113022959
[2023-03-16 17:48:10] FWD:OrmDynamicDataSetParam (count/ms/ns):     2040/83/83307547
[2023-03-16 17:48:10] FWD:OutputTableCopier (count/ms/ns):     1/0/71836
[2023-03-16 17:48:10] FWD:OutputTableHandleCopier (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:OutputDataSetCopier (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:OutputDataSetHandleCopier (count/ms/ns):     214/67/67285759
[2023-03-16 17:48:10] FWD:TMCommit (count/ms/ns):     12409/164/164526477
[2023-03-16 17:48:10] FWD:TMRollback (count/ms/ns):     1071/2/2132685
[2023-03-16 17:48:10] FWD:TMValidate (count/ms/ns):     15817/38/38735947
[2023-03-16 17:48:10] FWD:AppserverAddTableMetaData (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:ApperverGetJavaParameter (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:AppserverGetOutputParameter (count/ms/ns):     0/0/0
[2023-03-16 17:48:10] FWD:NetworkReads (count/total):     7889/1856436
[2023-03-16 17:48:10] FWD:NetworkWrites (count/total):     7889/2203077
[2023-03-16 17:48:10] FWD:WidgetAttrFlushes (count/total):     1/1
```

The 3 cases from note #6887-9 are:

```
[2023-03-16 17:48:10] FWD:OrmDataSetParam (count/ms/ns):     9135/113/113022959
[2023-03-16 17:48:10] FWD:OrmDynamicDataSetParam (count/ms/ns):     2040/83/83307547
[2023-03-16 17:48:10] FWD:OutputDataSetHandleCopier (count/ms/ns):     214/67/67285759
```

#### #13 - 04/05/2023 12:06 AM - Eric Faulhaber

Constantin Asofiei wrote:

> This is the entire JMX result for the large application:
> [...]
>
> The 3 cases from note #6887-9 are:
> [...]

These 3 cases look significantly better. It seems to me pushing more on this task probably will produce diminishing returns. Constantin, what is your opinion?

#### #14 - 04/05/2023 01:35 AM - Constantin Asofiei

We should profile and check how #6887-3 test behaves now with FWD first.

#### #15 - 05/11/2023 05:29 PM - Boris Schegolev

*- Status changed from WIP to Review*

#### #16 - 05/11/2023 05:48 PM - Eric Faulhaber

Boris, what is there to review?

#### #17 - 05/11/2023 05:51 PM - Boris Schegolev

I think we should close this task. I just didn't find a more appropriate status :)

#### #18 - 05/11/2023 06:06 PM - Eric Faulhaber

Constantin Asofiei wrote:

> We should profile and check how #6887-3 test behaves now with FWD first.

Boris, did you take a final pass at this? I didn't see any follow-up to this note.

Constantin, did something specific go into the runtime between March 15 (when Boris last profiled this) to April 5 (when you posted this note), that would make you think there may be an improvement?

#### #19 - 05/11/2023 06:12 PM - Boris Schegolev

Eric Faulhaber wrote:

> Boris, did you take a final pass at this?

I just did and I don't see any difference from what I measured the last time.

#### #20 - 05/12/2023 06:32 AM - Constantin Asofiei

Boris, did you run the tests in OpenEdge and in FWD? What is the difference in time?

In FWD, what does a trace profiling show for p2j.persist, org.h2, java.lang, java.util, p2j.util (excluding BlockManager)? I mean a break-down on low-level APIs being called, avoid wrapper APIs like internalProcedure which accumulate all the time, you need to drill down into it.

#### #21 - 05/16/2023 03:09 PM - Boris Schegolev

Constantin Asofiei wrote:

> Boris, did you run the tests in OpenEdge and in FWD? What is the difference in time?

Yes, I ran tests in both environments. The last time I had something like 200ms in OE and 350ms in FWD. But this runs on different HW, so the exact numbers should not be taken too seriously.

> In FWD, what does a trace profiling show for p2j.persist, org.h2, java.lang, java.util, p2j.util (excluding BlockManager)? I mean a break-down on low-level APIs being called, avoid wrapper APIs like internalProcedure which accumulate all the time, you need to drill down into it.

Again, I didn't see any 4s runs any more. Do we want to spend more time to optimize or this compares well enough already?

#### #22 - 01/19/2024 02:43 AM - Eric Faulhaber

*- Subject changed from improve performane of dataset/table[-handle] parameters to improve performance of dataset/table[-handle] parameters*

*- Assignee deleted (Boris Schegolev)*

Constantin, does it makes sense to do one more pass with JMX to see what the 3 cases which stood out earlier look like now? This is not the highest priority, given all the other work going on, but at some point pretty soon, we should figure out whether to prioritize this or close it out.