Base Language - Bug #2638

improve performance of user-def function calls converted to java method calls

08/13/2015 04:27 PM - Constantin Asofiei

Status:	Closed	**Start date:**	08/13/2015
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD		
Description			

History

#1 - 08/13/2015 04:28 PM - Constantin Asofiei

Greg,

On another thought, this is need only for user-defined function calls; so adding new parameters to the existing 2 BlockManager.function() APIs should be simpler - no need to expand the APIs with and without the call parameters, as these are mandatory.

Thanks,
Constantin

On 13.08.2015 18:33, Constantin Asofiei wrote:

> Greg,
>
> > ... or can it be placed in a single place in the generated function that is being called?
>
> I think it can be placed in the generated function (before the BlockManager.function API call), and bypass it (don't push the state) if there is a pending call via ControlFlowOps.
>
> Thanks,
> Constantin
>
> On 13.08.2015 18:28, Greg Shah wrote:
>
> > Constantin,
> >
> > Does the prepareCall() need to be in the calling function or can it be placed in a single place in the generated function that is being called? This would be a much smaller impact and seems possible. Theoretically we could even add a parameter to the BlockManager.function()... and other APIs too, but that seems more painful because of the heavy overloading that is already present.
> >
> > Greg
> >
> > On 08/13/2015 11:12 AM, Constantin Asofiei wrote:
> >
> > > Eric,
> > >
> > > The ProcedureManager.resolveClosestMethod is needed because converted code can emit a 4GL user-defined function call via a java method call or via ControlFlowOps APIs. So, in case when the call is converted to a java method call, we need to identify which 4GL function is being invoked and push its info. When the call is converted to a ControlFlowOps API, this is bypassed, as we already have all the info we need.
> > >
> > > I don't think there is a easy way beside fillInStackTrace() to automated this. But what we can do in the converted, for a function call, instead of this:
> > >
> > > callSome4GLFunction();
> > >
> > > we also emit another API call to prepare it:

ControlFlowOps.prepareCall(<info about what is being invoked>);
callSome4GLFunction();

This will eliminate the need for fillInStackTrace().

Or, the simplest alternative, add a configuration flag to always emit ControlFlowOps APIs.

Thanks,
Constantin

On 13.08.2015 17:50, Eric Faulhaber wrote:

> Hi Constantin,
>
> I've been doing a lot of performance profiling and improvements lately. One method that pops up as a severe problem in several use cases is ProcedureManager.resolveClosestMethod(Object, boolean, boolean). Due to its use of the very expensive fillInStackTrace() call (both in the Throwable c'tor and explicitly -- I've removed the latter already), it can really slow things down, sometimes considerably more than even database access, which usually is the slowest thing we do.
>
> resolveClosestMethod is called from BlockManager.checkJavaCall, which is used (a lot) by functionBlock and topLevelBlock. Can you think of any way to (a) avoid the need for resolveClosestMethod(); or (b) do what it does without using an exception to interrogate the Java stack? I know we use this technique in SecurityManager as well, but that does not seem to be called nearly as often.
>
> This is not a "drop everything" request, but if you could put some thought into it over the next days, I would appreciate it.
>
> Thanks,
> Eric

#2 - 08/13/2015 04:37 PM - Constantin Asofiei

Created task branch 2638a.

I've committed to rev 10921 the fix - it was pretty simple in the end, just bypass ProcedureManager.resolveClosestMethod if the legacy name is already known (i.e. emitted by conversion rules).

I'm running conversion and runtime testing now.

Eric: if you can, please leave a server conversion over night, with branch 2638a. The conversion will finish in ~1 hour - I'll let you know the state.

#3 - 08/13/2015 04:42 PM - Eric Faulhaber

Constantin Asofiei wrote:

> I've committed to rev 10921 the fix - it was pretty simple in the end, just bypass ProcedureManager.resolveClosestMethod if the legacy name is already known (i.e. emitted by conversion rules).

So, under what runtime conditions would resolveClosestMethod still be in use?

> Eric: if you can, please leave a server conversion over night, with branch 2638a.

OK.

#4 - 08/13/2015 04:48 PM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin Asofiei wrote:
>
> > I've committed to rev 10921 the fix - it was pretty simple in the end, just bypass ProcedureManager.resolveClosestMethod if the legacy name is already known (i.e. emitted by conversion rules).
>
> So, under what runtime conditions would resolveClosestMethod still be in use?

Conversion code emits java-style method calls only for a subset of legacy user-def functions calls - everything else is converted to ControlFlowOps APIs. And all the cases in this subset will pick up the legacy name emitted by the conversion rules, and resolveClosestMethod will not be used.

To be more specific:
- if there is only generated code, resolveClosestMethod will never be in use
- if there is hand-written java code, the dev writing the code might decide to invoke a procedure/function using java-style method call. And only in this case resolveClosestMethod will be in use.

#5 - 08/13/2015 04:50 PM - Eric Faulhaber

Constantin Asofiei wrote:

> To be more specific:
> - if there is only generated code, resolveClosestMethod will never be in use
> - if there is hand-written java code, the dev writing the code might decide to invoke a procedure/function using java-style method call. And only in this case resolveClosestMethod will be in use.

Nice. So, this should eliminate all the problematic cases I've profiled so far. Thanks very much for the quick turnaround!

#6 - 08/13/2015 04:53 PM - Eric Faulhaber

Code review 2638a/10921:

Changes look good. I'll report results of use with server after overnight conversion and testing tomorrow.

#7 - 08/13/2015 04:59 PM - Constantin Asofiei

I've added a small fix to rev 10922 - Added BlockManager.markPendingDynCall to ControlFlowOps.initializeExternalProcedure (used by initializing a proxy procedure invoked on an appserver); without it, P2J would have interpreted this call as a java-style method call.

Does not affect MAJIC.

#8 - 08/13/2015 06:08 PM - Constantin Asofiei

Eric: conversion passed, all changes are OK and expected.

Compilation fails because of hand-written java functions for which the fix is to add back the previous BlockManager APIs without the legacy function name as first param - see rev 10923 (again, they will be called only from hand-written code, conversion code does not use them).

If we decide to go another way with the hand written java code, I'll change it tomorrow.

#9 - 08/14/2015 02:07 PM - Constantin Asofiei

Branch 2638 rev 10923 passed runtime testing.

Rebased branch 2638 from trunk rev 10921 (new branch rev 10924).

#10 - 08/14/2015 02:52 PM - Eric Faulhaber

My testing of this update (i.e., branch rev. 10922) has gone well. I'm comfortable with you merging these changes into the trunk, though we have to settle on an approach for the breakage of the hand-written code. I'm OK with your suggested fix in note 8 (adding back the APIs upon which this hand-written code is dependent), since this is no worse than the original situation for this category of code.

#11 - 08/15/2015 03:19 AM - Constantin Asofiei

Rebased branch 2638 from trunk rev 10922 (new branch rev 10925).

#12 - 08/15/2015 03:22 AM - Constantin Asofiei

Branch 2638a rev 10925 was merged to trunk rev 10923 and archived. Notification email was sent.

#13 - 08/17/2015 07:57 AM - Greg Shah

- Status changed from WIP to Closed

#14 - 11/16/2016 12:06 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features