universität innsbruck

VU Scheduling Approaches in Distributed Systems.

Task 3 – "Definition of Optimization Constraints".

Student/Group:

- Simon Kleinfeld
- Tobias Kupek

General Feedback:

- Constraints: Correct solution, good program structure
- Evaluator-based solution: Also correct
- Approach comparison: Good comparison setup and good explanation. (Full disclosure :) : this is not what I expected. I have a decent idea why you see this behavior and we will discuss it in the lecture. But just as a take-away: the constraint-based solution will in general NOT BE faster per individual (it can, however, be faster to converge)).

Code Feedback:

- When counting the number of region constraint violations, you could also approach the problem from the side of the communications: If the src and the dest are in different regions, the message must be transmitted through links which are part of the cloud. So if you check whether a message is (a) connected to 2 secret tasks and (b) has at least one cloud link in its routing, you directly see whether there is a region constraint violation (you would actually also use a similar logic to formulate the region constraints).
- You could have a slightly more compact method for searching the data-dependent secret tasks by iterating directly over the successors (or predecessors) of tasks/messages instead of going over the edges (in this case, you are not checking any edge parameters, so you can directly go to the predecessor/successor node)
- Suuuuuuuuuuuper minor: Application and Architecture are vertex iterators, so that you can iterate over them directly instead of using .getVertices()
- Checks of the spec (successor number) and throwing exception if assumptions not satisfied — NICE
- Using a single constraint to forbid all secret task mappings onto cloud resources (instead of formulating a constraint per mapping) is interesting. Just to mentioned this: This is a thing which *could* have implications for the resolution speed of the constraint set, since the solver has a more difficult time learning implications (such as the fact that all these mappings are ALWAYS 0) when confronted with a longer equation. In this case, it does not make a difference since OpenDse preprocessed the constraint set and already takes out all fix variables before giving it to the solver.

- I like the way you are structuring your methods to actually have a dedicated method to formulate the constraints. Especially when formulating more complex constraint systems, this level of granularity makes your life much easier w.r.t. to testing and debugging. Just as a minor tip: You could add the actual constraint equation to the comment of the respective method, since this makes it easier to directly see which role the method plays in the overall constraint set.

- **Method visibility:** Just from personal experience, I would recommend you to use *protected* as the default visibility modifier for both methods and attributes. W.r.t. classes which are neither package neighbors nor children of the class you are writing, you get a very comparable degree of encapsulation as with *private*. At the same time, having set up your project right (i.e., in a way where the test directories mirror the structure of the main directories) you can directly access the protected methods/attributes in your tests, making testing much much easier (and enabling to test on a finer grain). E.g., in the Mapping Encoding in your project, it would be a good idea to write a test per method (e.g., testing that ONE contraint is generated exactly the right way), as opposed to creating a test on the level of a complete constraint set. The other argument is that by making methods or attributes private, you preventing users which may use your code outside of the project from accessing them, which can be the thing that you want in some cases, but in most cases will just force users of your code to apply ugly hacks. So, my recommendation for a rule of thumb is: everything is protected by default. You only make things public if you explicitly want them to be accessible by other classes. You only make things private if you explicitly want to prevent access from package neighbors and child classes. (Oooooohaaa. This text block became way tooo long :))

Summary:

As already for the previous tasks, very solid solution. I know that working with these constraint sets for the first time is super weird and requires a lot of patience. Well done :)

Best regards,

Fedor