zephyrus2

zephyrus2 is a tool that allows to compute the optimal configuration of applications to deploy. It is a completely new implementation of the previous Zephyrus tool developed within the Aeolus project (https://github.com/aeolus-project/zephyrus).

Zephyrus2 relies on SMT or CP solvers for the exploration of the search space. Moreover it comes with a completely new declarative specification language to define the desired configuration and now it supports binding preferences.

It is written in python. For more information please have a look at http://heim.ifi.uio.no/jacopom/publication/dblpconf__setta__abrahamcjkm16/

Docker Installation

zephyrus2 could be installed using the docker container technology (https://www.docker.com/) available for the majority of the operating systems. It can then be used by simply sending a post request to the server deployed by using docker.

The Docker image is availabe in Docker Hub. To install it please run the following commands.

```
sudo docker pull jacopomauro/zephyrus2
sudo docker run -d -p <PORT>:9001 --name zephyrus_container jacopomauro/zephyrus2
```

where <PORT> is the port used to use the functionalities of the service.

Assuming <JSON> is the json input file (please see below for more details about the input format), to obtain the desired configuration it is possible to perform the following post request.

```
curl -H "Content-Type: application/json" -X POST -d @<JSON> http://localhost:<PORT>/pr
```

To check if the service is responding it is possible to use the following get request. `curl http://localhost:<PORT>/health`

To clean up please lunch the following commands:

```
sudo docker stop zephyrus_container
sudo docker rm zephyrus_container
sudo docker rmi jacopomauro/zephyrus2
```

For more information, please see the Docker documentation at docs.docker.com

Some test examples file can be located in the zephyrus2/test subfolder (e.g., wordpress.json or wordpress_with_options.json).

Input Specification

Zephyrus2 supports for component and location specifications the JavaScript Object Notation (JSON) format. The formal schema of the JSON is defined in /specs/zephyrus_input_schema.json.

In the following we exploit a wordpress deployment to present a simple example on how to define location and components. The json file is available at /zephyrus2/tests/wordpress.json.

The general structure of the json file is the following one.

```
{
    "locations": { <locations objs, comma separated> },
    "components": { <components obj, comma separated> },
    "specification": <specification string>,
    "bind preferences": [ <binding preference strings, comma separated> ]
```

```
    "options": [ <option strings, comma separated > ].
}
```

The specification property is used to define the goal that Zephyrus has to achive using a declarative language.

The locations propoerties defines the kind of computational unitis (VMs) can be used to deploy the components that are defined using the components property.

The bind preferences are used to define optional preferences between the component connections.

Options is used to specify additional command line options. This should be a list of strings. Often, to run only Zephyrus and not the bind optimizer it is possible to specify the `options: [ ["--run-only-zephyrus" ]`

Location specification

To define for instance a c3 large Amazon virtual machines the JSON object to use is the following one.

```
"c3_large": {
    "num": 10,
    "resources": { "RAM": 3750 },
    "cost": 105
}
```

With the property `num` we define that there are 10 of this computational resources available, every one having a cost of 105 (property `cost`) and having 3.75 GB of RAM (property `resources`).

Resources can be arbitrary names (like `RAM` in the example) and they must take an integer value (e.g., `3750`). Components consume resources and Zephyrus guarantees that the cumulative sum of the components consumption deploy on a location does not overcome the provided value of resources available in the location.

Component specification

A component called `WordPress` can be defined as follows.

```
"WordPress": {
    "resources": { "RAM": 2000 },
    "requires": { "mysql_functionality": 2 },
    "provides": [ { "ports": [ "wp_backend_functionality" ], "num": 100 } ]
}
```

Here we are defining the component that consumes 2 GB or RAM (property `resources`).

This component to be installed requires the existence of a component providing the `mysql_functionality` (property `requires`). In particular it is also possible to specify the number of components providing the functionality `mysql_functionality`. In this case, for fault tolerance reasons we require the existence of 2 of such components.

With the property `provides` we instead specify the functionalities offered by the component to the other components. In this case Wordpress offers the functionality `wp_backend_functionality` to upto 100 other components that can be connected with it (if the component can offer its functionalities to an unbounded number of other components the keyword `num` can be left out).

Specification

Zephyrus2 uses a new specification language for deployment constraints. In the following we describe some main features of the language by means of simple examples, referring the interested reader to /zephyrus2/SpecificationGrammar/SpecificationGrammar.g4 for the formal grammar of the language.

A deployment constraint is a logical combination of comparisons between arithmetic expressions. Besides

integers, expressions may refer to component names representing the total number of deployed instances of a component. Location instances are identified by a location name followed by the instance index (starting at zero) in square brackets. A component name prefixed by a location instance stays for the number of component instances deployed on the given location instance. For example, the follow- ing formula requires the presence of at least one HTTP Load Balancer instance, and exactly one WordPress server instance on the second c3 large location instance.

```
HTTP_Load_Balancer > 0 and c3_large[1].WordPress = 1
```

For quantification and for building sum expressions, we use identifiers prefixed with a question mark as variables. Quantification and sum building can range over components, locations, or over components/ locations whose names match a given regular expression. Using such constraints, it is possible to express more elaborate properties such as the co-location or distribution of components, or limit the amount of components deployed on a given location. For example, the constraint

```
forall ?x in locations: ( ?x.WordPress > 0 impl ?x.MySQL > 0)
```

states that the presence of an instance of WordPress deployed on any location x implies the presence of an instance of MySQL deployed on the same location x. As another example, requiring the HTTP Load Balancer to be installed alone on a virtual machine can be done by requiring that if a Load Balancer is installed on a given location then the sum of the components installed on that location should be exactly 1.

```
forall ?x in locations: ( ?x.HTTP_Load_Balancer > 0 impl (sum ?y in components: ?x.?y)
```

For optimization, Zephyrus2 allows the user to express her preferences over valid configurations in the form of a list of arithmetic expressions whose values should be minimized in the given priority order. The keyword cost can be used to require the minimization of the total cost of the application. The following list specifies the metric to minimize first the total cost of the application and then the total number of components:

```
cost; ( sum ?x in components: ?x )
```

This is also the default metric used if the user does not specify her own preferences.

Binding preferences

Zephyrus2 allows the possibility to specify preferences over the connections or bindings between the components.

In the following we describe some simple examples, referring the interested reader to /zephyrus2/ bind_preference_grammar/BindPreferenceGrammar.g4 for the formal grammar of the preferences.

A preference may be either the string `"local"` or an arithmetic expression. The local preference is used to maximize the number of connections between the components deployed in the same location.

Arithmetic expression are used instead to capture more advanced preferences. These expressions are built by using as basic atoms integers and the predicate `bind(x,y,z)` that takes a value 1 if there is a connection between the component x and y where x provide to y the functionalities specified with the port name z, 0 otherwise. In order to instantiate the variables of the predicate bind, quantifiers and sum expressions may be used. As an example, the following query is used to maximize the number of Wordpress deployed on locations of type `c3_large` that are connected to all the Load Balancers deployed on locations locations of type `c3.*` via the port `wp_backend_functionality`

```
sum ?x of type Wordpress in `c3_large` :
 forall ?y of type LoadBalancer in `c3.*` :
    bind(?x,?y,wp_backend_functionality) = 1
```

In the first line we use the sum expression to match to the variable ?x all the Wordpress components hosted in a location whose name matches the regular expression `c3_large`. Similarly, in the second line we use the

forall expression to match to the variable ?y all the LoadBalancer deployed in a location whose name starts with the string c3. The forall expression is evaluated to 1 if, fixing the possible assignments of the variable ?y , the predicate `bind(?x,?y,wp_backend_functionality) = 1` is true. If instead there is an instance of LoadBalancer that is not connected to the Wordpress then the forall expression returns 0. Due to the fact that the first expression is a sum expression, the final behaviors of the preference is to maximize the number of instances of Wordpress deployed on c3_large instances that are connected with a LoadBalancer. With a forall, exists and sum expression, components can be filtered by their name. Regular expressions can be used to match the type of locations.

Options

When invoked, Zephyrus by defaults runs the zephyrus tool that computes the final configuration abstracting from the concrete bindings and then the bindings optimizer that tries to establish the bindings between the different components to maximize the user preferences.

To avoid the definition of the bindings, it is possible to invoke the tool to obtain only the abstract final configuration. To do so, following the input schema at /specs/zephyrus_input_schema.json, it is possible to add a property in the JSON input called `"options"` with the value `"--run-only-zephyrus"`.

The option property can also be used to pass other external parameters such as the possibility to disable the symmetry breaking constraint (`"--no-simmetry-breaking"`) or the selection of the backend solver to use (e.g., `"--solver","lex-gecode"` to use gecode or `"--solver","smt"` to use the Z3 SMT solver). Options can be combined by separating them with a comma (see, e.g., the test file zephyrus2/test/wordpress_with_options.json).

Output

Zephyrus outputs a JSON objects following the schema /specs/binding_optimizer_output_schema.json.