ICCP2018 Companion Paper: Use Cases Description

Flavia Micota and Mădălina Eraşcu and Daniela Zaharie
Department of Computer Science, West University of Timişoara
Institute e-Austria Timişoara,
Blvd. V. Pârvan 4, Timişoara, Romania;
E-mail: {flavia.micota, madalina.erascu, daniela.zaharie}@e-uvt.ro

June 23, 2018

Abstract

We present a detailed description of three applications: Secure Web Container, Wordpress and Oryx2.

1 Problem Description

For a self-contained presentation, in this section we recall the formalization of the problem.

The problem to be solved is to find a mapping (of application components to VMs) which: (1) satisfies the constraints induced by the interactions between components; (2) satisfies the hardware requirements of all components; and (3) minimize the purchasing price.

To describe the problem in a more formal way, let us consider a set of N interacting components, $\{C_1, C_2, \ldots, C_N\}$ to be assigned to a set of M virtual machines, $\{V_1, V_2, \ldots, V_M\}$. Each component C_i is characterized by a set of requirements concerning the hardware resources, e.g. number of CPUs (HC_i), memory (HM_i) - expressed in MB, storage (HS_i) - expressed in MB. The problem can be formulated as an integer programming problem, aiming to find the values of a set of binary variables, $a_{ik} \in \{0, 1\}$ for $i = \overline{1, N}$, $k = \overline{1, M}$ which are interpreted as follows:

$$a_{ik} = \begin{cases} 1 & \text{if } C_i \text{ is assigned to } V_k \\ 0 & \text{if } C_i \text{ is not assigned to } V_k \end{cases} \tag{1}$$

The overall search space is $\{0, 1\}^{M \times N}$ but the set of feasible solutions is smaller, being characterized by the constraints described in the following.

1.1 Overview of the constraints

As the study cases presented in Section 2 illustrate there are two main types of constraints: *hardware constraints* which refer to the hardware characteristics of the VM types offered by the Cloud Provider and *interaction constraints* that describe the interactions between components.

Hardware Constraints These constraints are of inequality type and specify that the amount of resources required by the components assigned to a VM does not overpass the VM characteristics, i.e.: are related to components hardware and have to be correlated to VMs offers from cloud providers:

$$\sum_{i=1}^{N} a_{ik} H_i^{res} \leq H_k^{res_{prov}}, \quad k = \overline{1, M} \tag{2}$$

where H^{res} is the required amount of resource (we considered in this paper only CPU, Memory and Storage capacity) and $H^{res_{prov}}$ is the corresponding characteristic of a VM included in an existing Cloud Provider offer.

Interaction Constraints These constraints correspond to various types of interactions identified in the case studies:

1. some components cannot be assigned to the same VM (*conflict*);
2. some components should be assigned to the same VM (*collocation*);
3. some components cannot be simultaneously deployed, not even on the same VM (*exclusive deployment*);
4. some components require functionalities offered by other components inducing constraints on the number of deployed instances (*require-provide*)

Besides the constraints expressing interactions between components, there are constraints referring to the number of deployed instances for a given component:

- some components should be deployed on all VMs (*full deployment*);
- the number of instances of a given component should check some lower and/or upper bound constraints (*deployment with bounded number of instances*).

1.2 Constraints Description

Conflicts This case corresponds to situations when there are conflictual components which cannot be deployed on the same VM. For example, in the case of the *Secure Web Container* application, we have the constraint that the Balancer component cannot be deployed on the same machine where any of components Apache or Nginx reside. In the general case we consider that all conflicts between components are encoded in a matrix, R, defined by: $R_{ij} = 1$ if C_i and C_j are conflictual components and $R_{ij} = 0$ otherwise. Thus these constraints can be described as a set of linear inequalities:

$$a_{ik} + a_{jk} \leq 1, \quad k = \overline{1, M}, \quad \text{for all } (i, j) \text{ such that } R_{ij} = 1.$$

Collocation This means that once a component is deployed on a VM an instance of the other component should be also deployed on the same VM. For example, in case of Oryx2 application, the components HDFS.DataNode and Spark.Worker must the collocated. In the general case, the collocation dependencies can be stored in a matrix D (i.e. $D_{ij} = 1$ if C_i and C_j should be collocated and $D_{ij} = 0$ otherwise) and the constraints can be described as a set of equalities:

$$a_{ik} = a_{jk}, \quad k = \overline{1, M}, \quad \text{for all } (i, j) \text{ such that } D_{ij} = 1$$

Exclusive deployment There are cases when two, or more, components cannot be simultaneously deployed. For instance, in the case of the Wordpress application, only one type of Balancer must be deployed even if there are two types of the Balancer components (HTTPLoadBalancer, DNSLoadBalancer). Such a constraint can be described as:

$$H(\sum_{k=1}^{M} a_{ik}) + H(\sum_{k=1}^{M} a_{jk}) = 1, \text{ for all } (i, j) \text{ such that } E_{ij} = 1$$

where E is the matrix describing the exclusive deployment relation (i.e. $E_{ij} = 1$ if C_i and C_j should be exclusively deployed and $E_{ij} = 0$ otherwise) and H is a Heaviside-like function ($H(u) = 1$ if $u > 0$ and $H(u) = 0$ if $u = 0$).

Require-Provide A special case of interaction between components is when one component requires some functionalities offered by other components as for instance in the case of Wordpress case study where a Wordpress component requires at least three instances of MySQL and one MySQL instance can serve at most 2 Wordpress instances. Such an interaction induces constraints on the number of instances corresponding to the interacting components as follows: *(1)* C_i requires (consumes) at least n_{ij} instances of C_j and *(2)* C_j can serve (provides) at most n_{ji} instances of C_i. This can be written as:

$$n_{ij} \sum_{k=1}^{M} a_{ik} \leq n_{ji} \sum_{k=1}^{M} a_{jk}, \quad n_{ij}, n_{ji} \in \mathbb{N}$$

Another related case appearing in the *Secure Web Container* case study is when for each set of n instances of component C_j a new instance of C_i should be deployed. This can be described as:

$$0 < n \sum_{k=1}^{M} a_{ik} - \sum_{k=1}^{M} a_{jk} \leq n, \quad n \in \mathbb{N}$$

Full Deployment As *Secure Web Container* and *Oryx2* case studies illustrate, there can be also cases when a component C_i must be deployed on all VMs (except on those which would induce conflicts). This can be expressed as:

$$\sum_{k=1}^{M} (a_{ik} + H(\sum_{j,R_{ij}=1} a_{jk})) = M$$

where R is the conflicts matrix and H is the Heaviside-like function defined above.

Deployment with bounded number of instances There are situations when the number of instances corresponding to a set of deployed components, $\overline{C}$, should be equal, greater or less than some values. For instance, in the case of Secure Web Container application, the total amount of instances of the Apache and Nginx components must be at least 3. These types of constraints can be described as follows:

$$\sum_{k=1}^{M} a_{ik} \ \langle \text{op} \rangle \ n, \quad \forall i \in \overline{C}, \quad \langle \text{op} \rangle \in \{=, \leq, \geq\}, \quad n \in \mathbb{N}$$

2 Case Studies

2.1 Secure Web Container

Secure Web Container is an important web security application. In [CBE+17], the authors have introduced a case study involving the provisioning of a *secure web container service* (Fig. 1) providing:

1. *resilience* to attacks and failures, by introducing redundancy and diversity techniques, and
2. protection from unauthorized and potentially dangerous accesses, by integrating proper *intrusion detection* tools.

Resilience was implemented by a set of different Web Container components and a Balancer component, which is responsible for dispatching web requests to the active web containers to ensure load balancing. In the simplest scenario, there are two Web Containers (e.g. Apache Tomcat[1] and Nginx). Intrusion detection is ensured by the generation of intrusion detection reports with a certain frequency. It was implemented by deploying an IDSAgent, to be installed on the resources to be protected, and an IDSServer, which collects data gathered by the IDSAgents and performs the detection activities. The constraints between application components are as follows.

- For Web resilience:
  1. The Balancer component cannot be deployed on the same machine where any of components Apache or Nginx reside; the Apache component cannot be deployed together with the Nginx component.

$$a_{1k} + a_{ik} \leq 1 \quad i \in \{2, 3\}, \quad k = \overline{1, M}$$
$$a_{2k} + a_{3k} \leq 1 \quad k = \overline{1, M}$$

  2. Exactly one component Balancer has to be instantiated.

$$\sum_{k=1}^{M} a_{1k} = 1$$

[1] http://tomcat.apache.org

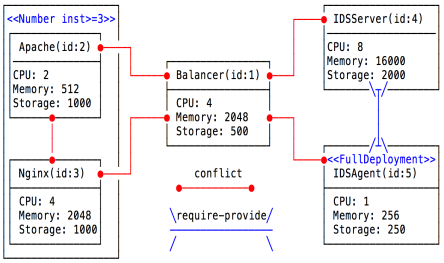

Figure 1: Secure Web Container Application

3. The total amount of instances of components Apache and Nginx must be at least 3 (level of redundancy).

$$\sum_{k=1}^{M} a_{2k} + \sum_{k=1}^{M} a_{3k} = 3, \quad k = \overline{1, M}$$

- For Web intrusion detection:

1. IDSServer components need exclusive use of machines.

$$a_{4k} + a_{ik} \leq 1, \quad i \in \{1, 2, 3, 5\}, \quad k = \overline{1, M}$$

2. There must be an IDSServer component additional instance every 10 IDSAgent component instances.

$$0 < 10 \sum_{k=1}^{M} a_{4k} - \sum_{k=1}^{M} a_{5k} \leq 10, \quad k = \overline{1, M}$$

3. One instance of IDSAgent must be allocated on every acquired machine except where an IDSServer or a Balancer are deployed.

$$\sum_{k=1}^{M} (a_{5k} + H(a_{1k}) + H(a_{4k})) = M$$

2.2 Wordpress

Wordpress open-source application is frequently used in creating websites, blogs and applications. We chose it in order to compare our approach to Zephyrus and Zephyrus2 deployment tools [CLT+14, ÁCJ+16]. In [CLT+14, ÁCJ+16], the authors present a high-load and fault tolerant Wordpress (Fig. 2) deployment scenario. The two characteristics are ensured by load balancing. One possibility is to balance load at the DNS level using servers like Bind[2]: multiple DNS requests to resolve the website name will result in different IPs from a given pool of machines, on each of which a separate Wordpress instance is running. Alternatively one can use as website entry point an HTTP reverse proxy capable of load balancing (and caching, for added benefit) such as Varnish. In both cases, Wordpress instances will need to be configured to connect to the same MySQL database, to avoid delivering inconsistent results to users. Also, having redundancy and balancing at the front-end level, one usually expects to have them also at the DBMS level. One way to achieve that is to use a MySQL cluster, and configure the Wordpress instances with multiple entry points to it.

In the deployment scenario considered by us, the following constraints must be fulfilled:

1. DNSLoadBalancer requires at least one instance of Wordpress and DNSLoadBalancer can serve at most 7 Wordpress instances.

$$\sum_{k=1}^{M} a_{3k} \leq 7 \sum_{k=1}^{M} a_{1k}, \quad k = \overline{1, M}$$

[2] https://www.isc.org/downloads/bind/

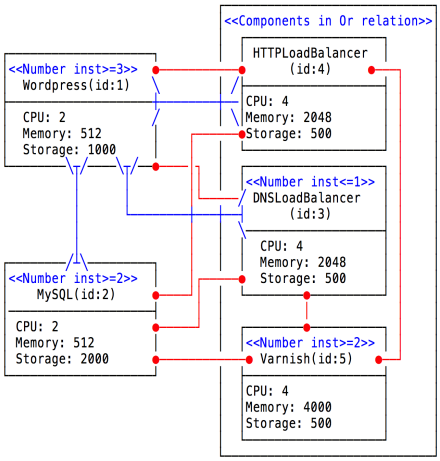

Figure 2: Wordpress Application

2. HTTPLoadBalancer requires at least one Wordpress instance and HTTPLoadBalancer can serve at most 3 Wordpress instances.

$$\sum_{k=1}^{M} a_{4k} \leq 3 \sum_{k=1}^{M} a_{1k}, \quad k = \overline{1, M}$$

3. Wordpress requires at least three instances of MySQL and MySQL can serve at most 2 Wordpress.

$$3 \sum_{k=1}^{M} a_{1k} \leq 2 \sum_{k=1}^{M} a_{2k}, \quad k = \overline{1, M}$$

4. Only one type of Balancer must be deployed; the Balancer components are HTTPLoadBalancer, DNSLoadBalancer and Varnish.

$$H(\sum_{k=1}^{M} a_{3k}) + H(\sum_{k=1}^{M} a_{4k}) = 1$$
$$H(\sum_{k=1}^{M} a_{3k}) + H(\sum_{k=1}^{M} a_{5k}) = 1$$

5. Since Varnish exhibits load balancing features should not be deployed with another type of Balancer. Moreover, Varnish and MySQL should not be deployed on the same VM because it is best practice to isolate the DBMS level of an application.

$$a_{5k} + a_{ik} \leq 1, \quad i \in \{2, 3, 4\}, \quad k = \overline{1, M}$$

6. If HTTPLoadBalancer is deployed then at least 2 instances of Varnish must be deployed too.

$$\sum_{k=1}^{M} a_{5k} \geq 2$$

7. At least 2 different entry points to the MySQL cluster.

$$\sum_{k=1}^{M} a_{2k} \geq 2$$

8. No more than 1 DNS server deployed in the administrative domain.

$$\sum_{k=1}^{M} a_{3k} \leq 1$$

9. Balancer components must be placed on a single VM, so they are considered to be in conflict with all the other components.

$$a_{3k} + a_{ik} \leq 1 \quad i \in \{1, 2, 5\}$$
$$a_{4k} + a_{ik} \leq 1 \quad i \in \{1, 2, 5\}$$

3 Oryx2

Oryx2 application is a realization of the lambda architecture, hence it is used in data analysis, and deploys the latest technologies such as Apache Spark[3] and Apache Kafka[4]. It has a significant number of components interacting with each other and is highly used in practical applications.

Oryx2[5] (Fig. 3) application is designed for data processing. It consists of several components which can be distributed over thousands of VMs in the case of a full deployment. Oryx2 has a lambda type architecture featuring the three tiers: speed, batch and serving. The main goal of Oryx2 is to take incoming data and use them to create and instantiate predictive models for various use-cases, e.g. movie recommendation. It is comprised of several technologies. Both the batch and serving layer are based on Apache Spark which in turn uses both Apache Yarn[6] for scheduling and Apache HDFS as a distributed file system. For a processing pipeline Oryx2 uses Apache Kafka with at least two topics; one for incoming data and one for model update. Apache Zookeeper[7] is used by Kafka for broker coordination. All of the aforementioned technologies have subservices with a minimum system requirement and recommended deployment as of Fig. 3.

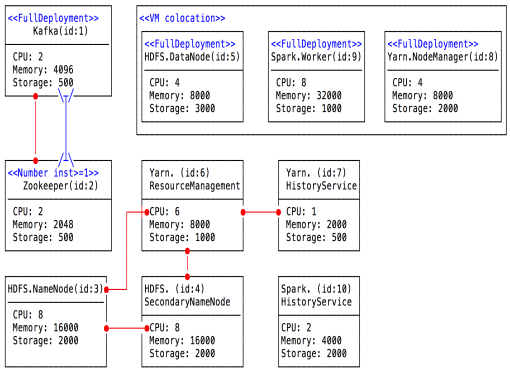

Figure 3: Oryx2 Application

The restrictions between the components are in the following:

1. Components HDFS.DataNode and Spark.Worker must the deployed on the same VM. In this scenario, we also collocated Yarn.NodeManager because we used Yarn as a scheduler for Spark jobs.

$$a_{ik} = a_{9k}, \quad i \in \{5, 8\}, \quad k = \overline{1, M}$$

2. Components Kafka and Zookeeper, HDFS.NameNode and HDFS.SecondaryNameNode, YARN.ResourceManagement and HDFS.NameNode, HDFS.SecondaryNameNode, YARN.HistoryService are, respectively in conflict, that is, they must not be placed on the same VM.

$$a_{1k} + a_{2k} \leq 1$$
$$a_{3k} + a_{4k} \leq 1$$
$$a_{6k} + a_{ik} \leq 1, \quad i \in \{3, 4, 7\}, \quad k = \overline{1, M}$$

[3] https://spark.apache.org/
[4] https://kafka.apache.org/
[5] http://oryx.io
[6] http://hadoop.apache.org/
[7] https://zookeeper.apache.org/

3. Components HDFS.DataNode, YARN.NodeManager and Spark.Worker must be deployed on all VMs except those hosting conflicting components.

$$\sum_{k=1}^{M}(a_{ik} + H(a_{5k}) + H(a_{8k}) + H(a_{9k})) = M$$

4. In our deployment, we consider that one instance of Kafka there must be deployed exactly 2 instances of Zookeeper. There can be situations, however, when more Zookeeper are deployed for higher resilience.

$$\sum_{k=1}^{M} a_{1k} = 2\sum_{k=1}^{M} a_{2k}$$

5. A single instance of YARN.HistoryService, respectively Spark.HistoryService should be deployed.

$$\sum_{k=1}^{M} a_{ik} = 1, \quad i \in \{7, 10\}, \quad k = \overline{1, M}$$

6. At least one instance of Zookeeper should be deployed.

$$\sum_{k=1}^{M} a_{2k} \geq 1$$

References

[ÁCJ+16] Erika Ábrahám, Florian Corzilius, Einar Broch Johnsen, Gereon Kremer, and Jacopo Mauro. Zephyrus2: On the fly deployment optimization using SMT and CP technologies. In *Dependable Software Engineering: Theories, Tools, and Applications - Second International Symposium, SETTA 2016, Beijing, China, November 9-11, 2016, Proceedings*, pages 229–245, 2016.

[CBE+17] Valentina Casola, Alessandra De Benedictis, Madalina Erascu, Jolanda Modic, and Massimiliano Rak. Automatically enforcing security slas in the cloud. *IEEE Trans. Services Computing*, 10(5):741–755, 2017.

[CLT+14] Roberto Di Cosmo, Michael Lienhardt, Ralf Treinen, Stefano Zacchiroli, Jakub Zwolakowski, Antoine Eiche, and Alexis Agahi. Automated synthesis and deployment of cloud applications. In *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, pages 211–222, 2014.