ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ
ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ
ΒΙΟΙΑΤΡΙΚΗ»

Μετακίνηση Εικονικών Μηχανών και Αντιγραφή Δεδομένων σε Κέντρα Δεδομένων που Βρίσκονται στις Παρυφές Κυψελωτών Δικτύων

Αρετή Μπαχτσεβάνη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Υπεύθυνος
Αθανάσιος Λουκόπουλος

Λαμία, 2017

UNIVERCITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL

OF BIOMEDICINE

Virtual Machine Migration and Data Replication in Mobile Edge-Computing Datacenters

Areti Bachtsevani

Master thesis

Thanasis Loukopoulos

Lamia

2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ

«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ»

Μετακίνηση Εικονικών Μηχανών και Αντιγραφή Δεδομένων σε Κέντρα Δεδομένων που Βρίσκονται στις Παρυφές Κυψελωτών Δικτύων

Αρετή Μπαχτσεβάνη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Αθανάσιος Λουκόπουλος

Λαμία, 2017

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο «Μετακίνηση Εικονικών Μηχανών και Αντιγραφή Δεδομένων σε Κέντρα Δεδομένων που Βρίσκονται στις Παρυφές Κυψελωτών Δικτύων» αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Μέρη της παρούσας εργασίας έχουν δημοσιευθεί στην [21].

Η ΔΗΛΟΥΣΑ

Ημερομηνία

Υπογραφή

Μετακίνηση Εικονικών Μηχανών και Αντιγραφή Δεδομένων σε Κέντρα Δεδομένων που Βρίσκονται στις Παρυφές Κυψελωτών Δικτύων

Αρετη Μπαχτσεβάνη

Τριμελής Επιτροπή:

Ονοματεπώνυμο, Αθανάσιος Λουκόπουλος

Ονοματεπώνυμο, Γεώργιος Σταμούλης

Ονοματεπώνυμο, Κωνσταντίνος Αντωνής

Abstract

Several virtual machine (VM) placement algorithms have been proposed and studied in the literature with various scopes such as server consolidation or network cost minimization. In most cases, decisions on VM migrations are taken without factoring in directly the data access cost by VMs. In this thesis, I investigate the use of data replication in conjunction with the VM assignment problem and target on developing algorithms that decide both on which data should be replicated where and which VM must be migrated so as to minimize the network overhead among traditional cloud and mobile cloud systems. I discuss both the un-capacitated case and the more realistic case whereby datacenters (for the traditional cloud case) and micro-datacenters (for the mobile cloud case) have limited storage and computing capacity. I propose an algorithm based on hyper-graph partitioning to solve the aforementioned problem in an optimal way regarding the unconstrained case and extend it to capture storage and computing capacity constraints. Experimental evaluation shows that the proposed algorithm yields up to 53% network overhead reduction when compared to state-of-the-art algorithms found in the literature. Parts of this thesis were published in [21].

Περιληψη

Έχουν προταθεί στη βιβλιογραφία αρκετοί αλγόριθμοι τοποθέτησης που έχουν σαν στόχο την ελαχιστοποίηση ενέργειας, τη μείωση του κόστους δικτύου, κτλ. Στις περισσότερες περιπτώσεις, οι αποφάσεις για την μετακίνηση εικονικών μηχανών λαμβάνονται χωρίς να υπολογίζεται άμεσα το κόστος της πρόσβασης δεδομένων από τις εικονικές μηχανές. Σε αυτή την εργασία, διερευνώ την χρήση αντιγραφής δεδομένων σε συνδυασμό με το πρόβλημα ανάθεσης εικονικών μηχανών και στοχεύω στην ανάπτυξη αλγορίθμων που αποφασίζουν που και ποια δεδομένα θα πρέπει να αντιγραφούν καθώς και ποιες εικονικές μηχανές θα πρέπει να μετακινηθούν για να ελαχιστοποιηθεί ο φόρτος δικτύου μεταξύ παραδοσιακών συστημάτων υπολογιστικών νεφών και συστημάτων υπολογιστικών νεφών για κινητές συσκευές. Διερευνάται τόσο η περίπωση όπου δεν υπάρχουν περιορισμοί στην χωρητικότητα των κέντρων δεδομένων όσο και η πιο ρεαλιστική περίπωση όπου τα παραδοσιακά κέντρα δεδομένων καια τα κέντρα δεδομένων για κινητές συσκευές έχουν περιορισμένη αποθηκευτική και υπολογιστική χωρητικότητα. Προτείνω έναν αλγόριθμο που βασίζεται σε διαμερισμό υπεργράφων για να επιλύσω το προαναφερθέν πρόβλημα με βέλτιστο τρόπο όσον αφορά στην περίπτωση που δεν υπάρχουν περιορισμοί στην χωρητικτα και τον επεκτείνω έτσι ώστε να επιλύει το πρόβλημα όταν υπάρχουν περιορισμοί αποθηκευτικής και υπολογιστικής χωρητικότητας Η πειραματική αξιολόγηση δείχνει ότι ο προτεινόμενος αλγόριθμος επιτυγχάνει μέχρι και 53% μείωση στο φόρτο δικτύου σε σύγκριση με αλγορίθμους που έχουν προταθεί στη βιβλιογραφία. Μέρη της παρούσας εργασίας έχουν δημοσιεθεί στην [21].

Table of Contents

1 Introduction and background

Traditional cloud computing has received a lot of attention from the research community during the last decade. There have been many scientific projects generating enormous amounts of data ranging from a few dozen terabytes to petabytes. Such is the case with the Compact Muon Solenoid experiment [35] at CERN, the Human Genome Project [37], and the Human Brain Project [38]. Besides the insatiable demands of such scientific projects in data storage and management, there are also voracious demands for computing resources by a huge number of scientific applications that need to process the generated data.

The above voracious demands in both data and computational resources have urged researchers to deploy various systems and methodologies for traditional cloud systems. For example, many data- and compute-intensive middleware solutions do exist in the literature, namely, Hadoop [34], Apache HAMA [33], StorkCloud [10], to name a few. A general approach of those initiatives is to move computations close to data. The above is corroborated by the Hadoop community [34] stating the following: "*A computation requested by an application is much more efficient if it is executed near the data it operates on. HDFS provides interfaces for applications to move themselves closer where the data is located.*"

While moving computations towards data may be the best approach for traditional cloud applications, it is not always a winning strategy for applications located in both traditional cloud and mobile cloud, where mobile cloud consists of micro-datacenters located at some base stations of cellular networks. Particularly, in mobile cloud systems (also called mobile edge computing systems), both data and computations are

pushed to the logical extremes of the underlying network such that mobile subscribers can experience less delay against having data and computations at centralized nodes.

In this thesis, I address the problem of simultaneously taking VM placement and replica placement decisions in tree-structured networks to reduce the overall network overhead incurred due to the communication dependencies between VMs and data. In that way, the network overhead between traditional and mobile cloud is optimized. Since the communication between mobile cloud and the traditional cloud is considered slow, by targeting the aforementioned optimization goal I also optimize the delay experienced by mobile subscribers. The reason that we focus in tree-structured networks is justified by the fact that the traditional cloud can be seen as the root of a two-level tree, where micro-datacenters have one-hop access to traditional cloud (i.e., traditional datacenters). Because mobile edge computing is still at its infancy, we can also safely assume that as this technology becomes mature, deep micro-datacenter hierarchies will be established. Due to scalability reasons, we assume that fat-tree networks will prevail over other network structures.

The contributions of this thesis are summarized as follows:

- We introduce the problem of jointly performing data replication and virtual machine migrations to minimize network overhead in edge computing systems.
- We propose an algorithm based on hyper-graph partitioning to solve the problem when no capacity constraints are taken into account. The algorithm is extended to take into account capacity constraints.
- We formally prove that when no capacity constraints are taken into account, the proposed algorithm always results in an optimal solution.

I provide an experimental evaluation to show the performance of the proposed algorithms. It must be noted that parts of this thesis were published in [21].

2 System Model and Problem Formulation

2.1 System model

The application is structured as a general graph whereby nodes depict VMs and data chunks. The total number of VMs and data chunks within the graph is V and D, respectively. Let v_i and d_j denote the i^{th} VM and j^{th} data chunk, respectively. The size of j^{th} data object and i^{th} VM (measured in bytes) is captured by $\delta(d_j)$ and $\delta(v_i)$, respectively. The computing requirements of i^{th} VM is captured by $r(v_i)$. The volume of data exchanged between VMs as well as the data access cost is encoded by a matrix $C \in Z_{\geq 0}^{(V+D)\times(V+D)}$. Specifically, there are three cases for an entry c_{ij} of C: (i) when both i and j are less than V+1, then c_{ij} captures the data exchanged between i^{th} and j^{th} VM; (ii) when i is less than or equal to V (or more than V, respectively) and j is more than V (or less than V+1, respectively), then c_{ij} captures the volume of data accessed from v_i regarding $d_{j\text{-}V}$ (or c_{ij} captures the volume of data accessed v_j regarding $d_{i\text{-}V}$, respectively); and (iii) when both i and j are more than V, then c_{ij} is always zero because there is no case where a data object accesses another data object. We must note that C is symmetric.

The network is structured as a tree, with S being the number of clusters within the system. Let s_x signify the x^{th} cluster within the system, while r_x denote the total computing resources of s_x. The distance between clusters is captured by a matrix $W \in Z_{\geq 0}^{S\times S}$. Each entry of W is captured by w_{xy} denoting the distance (measured in hops) between s_x and s_y.

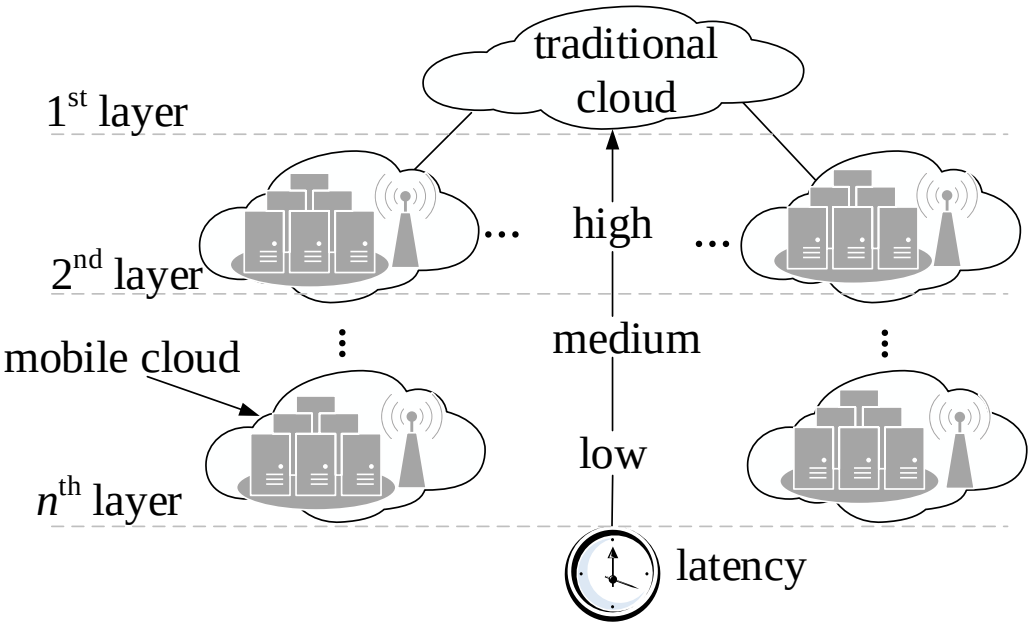

Figure 1. Tree-structured network.

The network is structured as a tree (the interested reader is referred to [20]), with S being the number of datacenters within the system (including micro-datacenters and traditional datacenters).). Let s_x signify the x^{th} datacenter (micro-datacenter or traditional datacenter) within the system, while r_x denote the total computing resources of s_x. Such a network is shown in Fig. 1, with the rooted node denoting the traditional datacenters; while the rest nodes signifying micro-datacenters. As shown in the figure, the latency increases when moving from the edges of the network towards the centralized nodes. The distance between datacenters is captured by a matrix $W \in Z_{\geq 0}^{S \times S}$. Each entry of W is captured by w_{xy} denoting the distance (measured in hops) between s_x and s_y.

Consider an example of a multi-tier service existing in a traditional cloud. We should decide which components of that service may be placed towards the edges of the network such that to decrease the network overhead (and thus latency) incurred within the system when clients using that service.

2.2 Problem Formulation

Before proceeding to the problem formulation, we will extend the notations. The placement of VMs and data on datacenters is captured by an $(V+D)\times S$ matrix denoted by F. Let f_{ix} whether the x^{th} datacenter hosts the i^{th} object (f_{ix} equals 1) or otherwise (f_{ix} equals zero). If i is less than or equal to V, then the i^{th} object represents a VM, otherwise a data object. Given a placement F, Eq. 1 captures the network overhead incurred within the system due to the communication between VMs. Let $R(k)$ be the set of datacenters hosting replicas of k^{th} data object. Eq. 2 denotes the minimum distance among the distances between s_x and the datacenters hosting a replica of k^{th} data object. The network overhead incurred due to the data accesses of the VMs is signified by Eq. 3.

The datacenter hosting the primary replica of k^{th} data object is denoted by $P(k)$. The variable φ_{kx} (captured by Eq. 4) denotes the cost of replicating k^{th} data onto x^{th} datacenter. Specifically, Eq. 4 states that x^{th} datacenter gets k^{th} data object from the nearest datacenter hosting or is to host k^{th} data object, which is located on the path between s_x and $s_{P(k)}$ (including $s_{P(k)}$). Eq. 5 captures the network overhead incurred due to the replication of all of the data objects. Eq. 6 signifies the total network overhead due to the needs of VMs to access data. Eq. 7 and Eq. 8 represent the storage and computing capacity constraints, respectively.

The problem is formally stated as follows: *Given a set of VMs along with their computing requirements, a set of data objects along with their initial assignment on datacenters, and a set of datacenters along with their computing capacities, find maximize (6) subject to (7) and (8).*

It is well-known that the problem becomes NP-Complete when considering general structured networks as well as capacitated data-centers [25], [26].

$$\Delta 1(F) = \sum_{i=1}^{V}\sum_{j>i}^{V}\sum_{x=1}^{S}\sum_{y=1}^{S} c_{ij} \times f_{ix} \times f_{jy} \times w_{xy} \qquad (1)$$

$$\lambda_{kx}(F) = \min_{\forall y \in R(k)} w_{xy} \quad (2)$$

$$\Delta 2(F) = \sum_{i=1}^{V} \sum_{k=V+1}^{V+D} \sum_{x=1}^{S} \sum_{y=1}^{S} c_{ij} \times f_{ix} \times \lambda_{kx}(F) \quad (3)$$

$$\varphi_{kx}(F) = \min_{\forall y \in R(k), y \neq x} w_{xy} \mid w_{yP(k)} \leq w_{xP(k)} \quad (4)$$

$$\Delta 3(F) = \sum_{k=V+1}^{V+D} \sum_{x=1}^{S} f_{kx} \times \varphi_{kx}(F) | x \neq P(k) \quad (5)$$

$$\Delta(F) = \Delta 1(F) + \Delta 2(F) + \Delta 3(F) \quad (6)$$

$$\sum_{j=1}^{D} f_{V+j,x} \times \delta(d_j) \leq \delta(s_x), 1 \leq x \leq S \quad (7)$$

$$\sum_{i=1}^{V} f_{ix} \times r(v_i) \leq r(s_x), 1 \leq x \leq S \quad (8)$$

3 Hyper-Graph Partitioning Algorithm (HPA): The Two Datacenter Case

3.1 Problem Reduction

Theorem 1. *The problem addressed in this thesis for the two-datacenter case is reduced to the hyper-graph bi-partitioning problem when a data object is only fully accessed by VMs.*

Proof. The input to our problem is an application graph comprising of VMs and data as discussed in Section 2.1, as well as a network of two datacenters. The transformation of such an application graph into a hyper-graph takes place as follows. A hyper-edge is created for each data object if the latter is (fully) accessed by more than one VM. The weight of such a hyper-edge equals the size of the corresponding data object and signifies the overhead that will be incurred within the network for the replication of the corresponding data object if the VMs belonging to the respective hyper-edge are not co-located with the respective data object. The communication dependencies between a pair of VMs are captured by a regular edge, with its weight equaling the volume of data exchanged between the respective VMs. We must also note that when a data object is (fully) accessed by only one VM, then this is captured by a regular edge, with its weight equaling the size of the corresponding object. When cutting a regular edge or a hyper-edge, then the network overhead is burdened by the weight of the respective edge. Therefore, by partitioning (cutting) the graph into two parts, we result in a network overhead equaling the weight of the cut in question. Note that the cut may consist of a set of hyper-edges and/or regular edges. Such a partition/cut encodes the VM assignment and data replication onto two datacenters. This is the well-known hyper-graph bi-partitioning problem [31]. Therefore, solving the aforementioned hyper-graph bi-partitioning problem is

equivalent to finding the minimum cut of the respective hyper-graph; and thus resulting in an assignment of VMs as well as a replication of data objects onto two datacenters with the minimum network overhead. ■

The aforementioned is illustrated through the following example. Consider the application shown in Fig. 2. The application consists of four VMs that fully access two different data objects (d_1, d_2). An edge between a VM v_i and data object d_j denotes that v_i needs to fully access d_j. An edge between two VMs represents the exchange of data between the respective VMs, with the weight of the respective edge signifying the volume of data being exchanged. Consider also the network shown in Fig. 3 consisting of two datacenters (s_1, s_2), with s_1 hosting d_1, while s_2 hosting d_2.

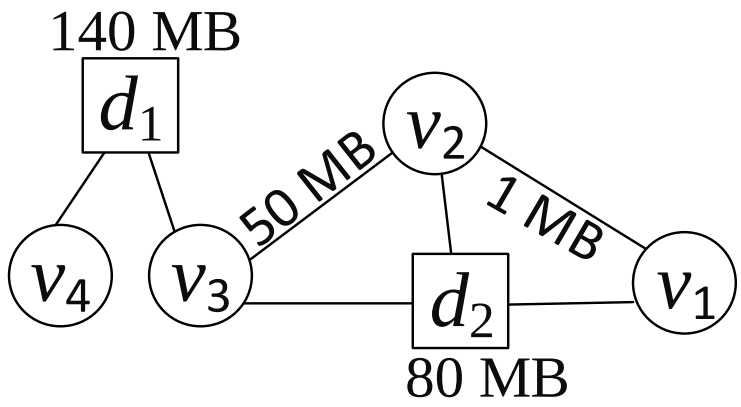

Figure 2. Application and data graph

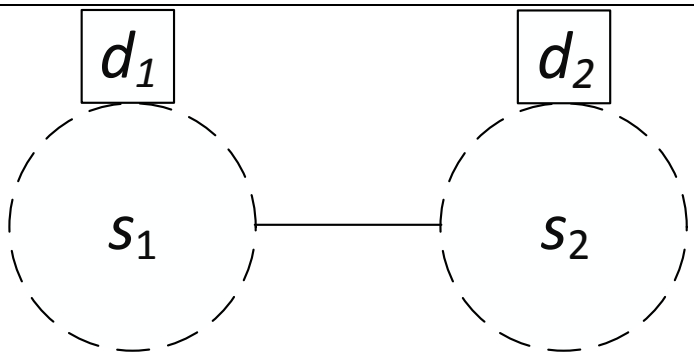

Figure 3. Network graph

In the sequel, we find all of the hyper-edges within the application graph. The hyper-edges are shown in Fig. 4. For each data object, we create a hyper-edge containing the respective data object, as well as the VMs needing access to the data object in question. As stated in the preceding text, the weight of a hyper-edge signifies the network overhead that will be incurred within the network if the VMs belonging to the respective hyper-edge are not co-located with the data object in question. For example, when v_3, v_4, and d_1 are not co-located, then the network will be burdened with a network overhead of 140 MB. For instance, consider the following three cases: **(a)** both v_3 and v_4 are placed on s_2 and the dashed hyper-edge is split, with the network being burdened by 140 MB due to the replication of d_1 on s_2. Note that without replication and assuming that v_3 and v_4 access d_1 at different points in time, then v_3 and v_4 must separately access data object through a network connection incurring 280 MB (140 + 140); **(b)** v_3 is placed on s_2, while v_4 on s_1, with the dashed hyper-edge being split. Therefore, we can either replicate d_1 on s_2, or we can assume that v_3 remotely accesses d_3 through a network connection, with the network being burdened in both cases by 140 MB; **(c)** both v_3 and v_4 are placed on s_1, whereby there will be no network overhead because both v_3 and v_4 access locally d_1.

As a result, to minimize the network overhead, we must bi-partition the hyper-graph shown in Fig. 4. The minimum cut is achieved by assigning all of the VMs on s_1, and then replicating d_1 on s_1. The total network overhead of the above assignment becomes 80 MB. The above result is explained by the fact that the solid hyper-edge is split, thus incurring 80 MB within the network due to the replication of d_2 on s_1. The final VM placement and data replicas are shown in Fig. 5.

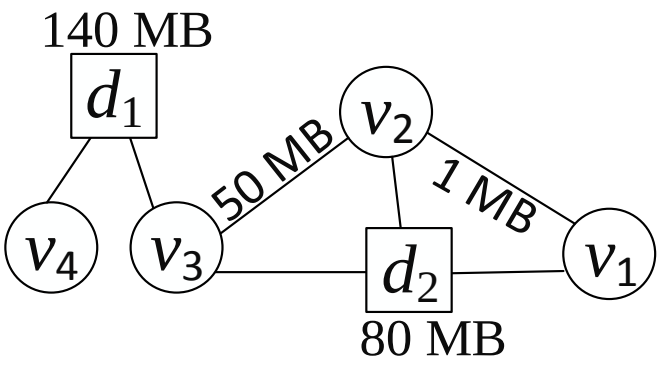

Figure 2. Application and data graph

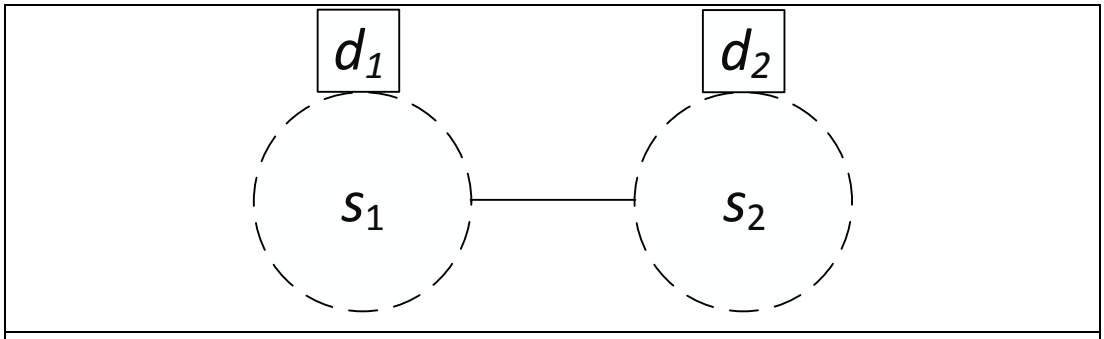

Figure 3. Network graph

In the sequel, we find all of the hyper-edges within the application graph. The hyper-edges are shown in Fig. 4. For each data object, we create a hyper-edge containing the respective data object, as well as the VMs needing access to the data object in question. As stated in the preceding text, the weight of a hyper-edge signifies the network overhead that will be incurred within the network if the VMs belonging to the respective hyper-edge are not co-located with the data object in question. For example, when v_3, v_4, and d_1 are not co-located, then the network will be burdened with a network overhead of 140 MB. For instance, consider the following three cases: **(a)** both v_3 and v_4 are placed on s_2 and the dashed hyper-edge is split, with the network being burdened by 140 MB due to the replication of d_1 on s_2. Note that without replication and assuming that v_3 and v_4 access d_1 at different points in time, then v_3 and v_4 must separately access data object through a network connection incurring 280 MB (140 + 140); **(b)** v_3 is placed on s_2, while v_4 on s_1, with the dashed hyper-edge being split. Therefore, we can either replicate d_1 on s_2, or we can assume that v_3 remotely accesses d_3 through a network connection, with the network being burdened in both cases by 140 MB; **(c)** both v_3 and v_4 are placed on s_1, whereby there will be no network overhead because both v_3 and v_4 access locally d_1.

As a result, to minimize the network overhead, we must bi-partition the hyper-graph shown in Fig. 4. The minimum cut is achieved by assigning all of the VMs on s_1, and then replicating d_1 on s_1. The total network overhead of the above assignment becomes 80 MB. The above result is explained by the fact that the solid hyper-edge is split, thus incurring 80 MB within the network due to the replication of d_2 on s_1. The final VM placement and data replicas are shown in Fig. 5.

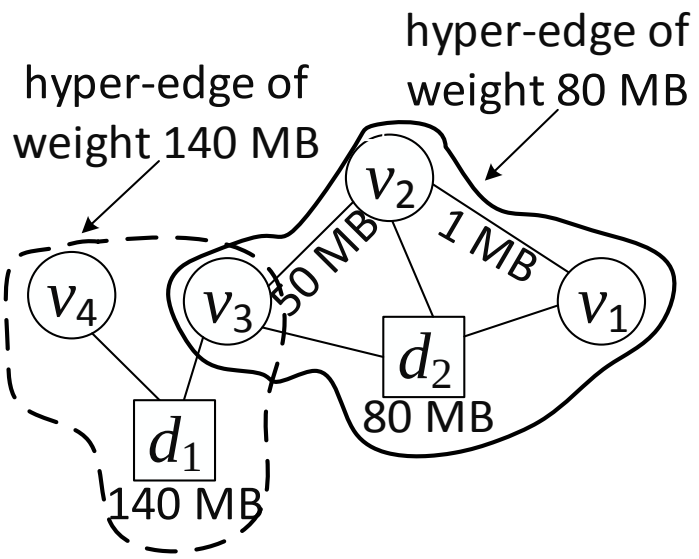

Figure 4. Hyper-edges

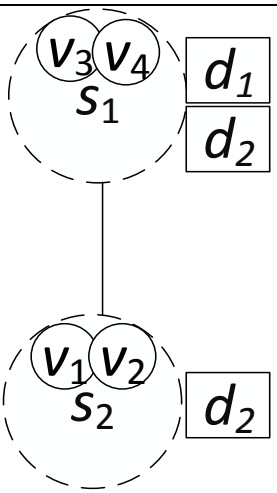

Figure 5. Final VM placement and data replicas

3.2 HPA Functionality When Data Objects are Only Fully Accessed by VMs

In this section, we consider that the VMs have not been assigned to any datacenter within the system. Therefore, we examine the initial VM placement as well as the data object replicas, assuming that data objects are only fully accessed by VMs. The functionality of the algorithm is split into the following phases.

3.2.1 Hyper-edge transformation into a set of regular edges

Consider a hyper-edge e of weight ω along with n vertices (u_1, ..., u_n) participating in the respective hyper-edge, then the hyper-edge can be transformed into a set of regular edges as follows (see [12]): **(a)** we add two auxiliary vertices (u_x and u_y) as well as a bridging edge of weight ω between them; and **(b)** for each vertex u_i belonging to e, we add two directed edges of infinite weight. The first one originates from u_i and ends on u_x, while the second one originates from u_y and ends on u_i.

An example is shown below where a hyper-edge consisting of five vertices, with its weight equaling 80 (Fig. 6), is transformed into a set of regular edges (Fig. 7). As can be seen, the weight of bridging edge between u_x and u_y equals 80, while the weight of the remaining edges equals infinity.

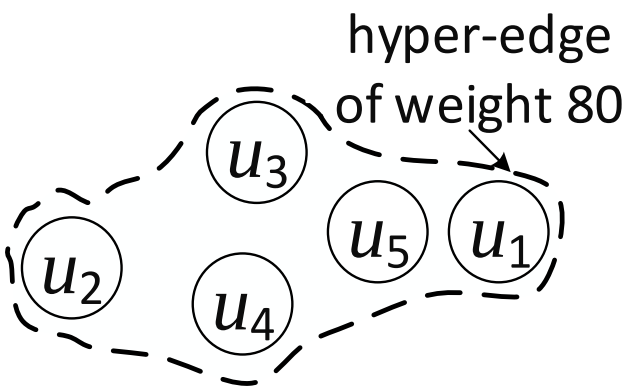

Figure 6. Hyper-edge of weight 80 with five vertices

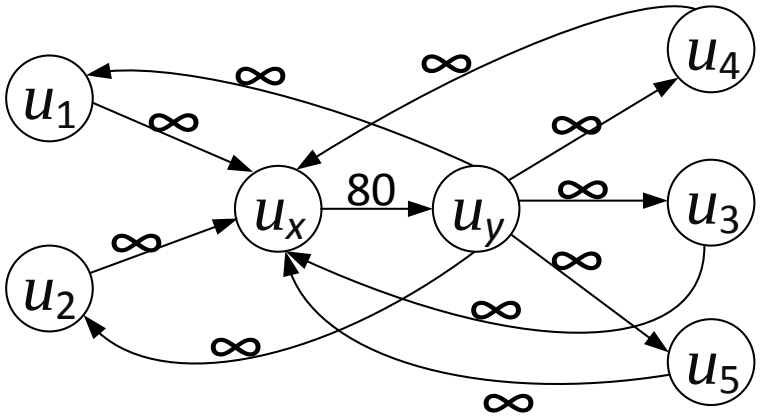

Figure 7. Hyper-edge transformation

3.2.2 Applying minimum cut algorithm

Given an application that needs access to some data objects hosted by two datacenters, then the VM placement and data replication problem is solved by employing the technique solving the {s-t} minimum cut problem as follows: **(i)** add two vertices s_1 and s_2, representing the corresponding two datacenters; **(ii)** for each hyper-edge perform the transformation explained in the previous section, with the bridging edge equaling the size of the data object (d_i) belonging to the respective hyper-edge. Note that instead of using an abstract name for the auxiliary vertex u_x (u_y, respectively) we use the name of the data object d_i ($d_{i'}$, respectively) belonging to the respective edge; **(iii)** add a directed edge of infinite weight originating from d_i's hosting datacenter and ending on d_i, as well as an edge of infinite weight originating from $d_{i'}$ and ending on d_i's hosting datacenter; **(iv)** if there is any regular edge within the application graph, then add this edge as is to the new resultant graph called minimum-cut graph; **(v)** we apply on the resultant graph the maximum flow minimum cut algorithm for undirected edges [17], considering that s_1 and s_2 play the role of source and terminal, respectively. Note that any undirected edge is transformed into a pair of directed ones; **(vi)** the set of vertices that is reachable from s_1 in the resulting residual network are assigned onto s_1, while the rest ones are assigned onto s_2. Specifically, when a vertex represents a VM, which is reachable from s_1 then the corresponding VM is assigned on s_1. On the other hand, when a vertex represents a data object (let d_k) we discern the following cases: **(a)** the edge between d_k and $d_{k'}$ belong to the minimum cut, with d_k being replicated on s_1 (s_2, respectively) given that d_k is hosted by s_2 (s_1, respectively); and **(b)**the edge between d_k and $d_{k'}$ does not belong to the minimum cut, and no replication takes place regarding d_k. Note that when d_k is replicated from s_1, with $d_{k'}$ being not reachable from s_1; then d_k is replicated on s_2. The main concept behind the minimum cut maximum flow algorithm is to obtain individual augmenting paths that can be used to increment an existing flow. An augmenting path is a directed path from source to sink that increases the existing flow.

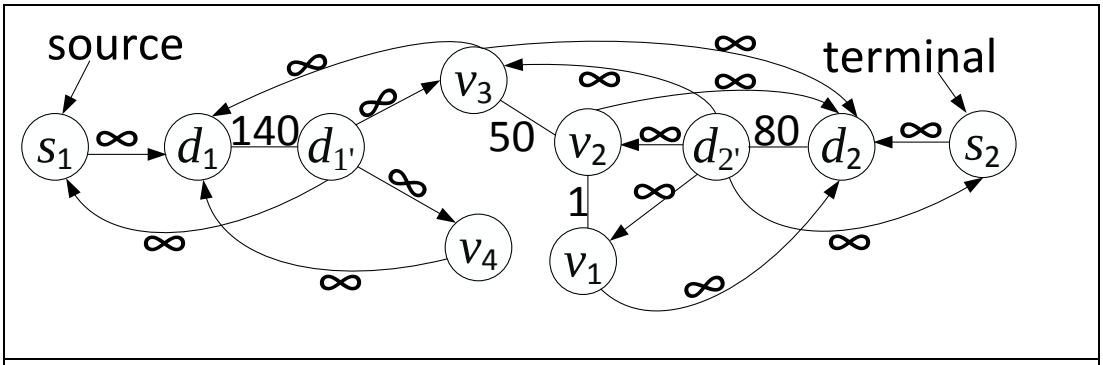

Figure 8. Minimum cut graph

Subsequently, we give an example to illustrate the aforementioned process. Consider an application that needs to access two data objects (d_1 and d_2). The application structure and the access of data objects by VMs are shown in Fig. 4. Note that we assume that the data objects are fully accessed by the VMs having an edge on them. As can be seen, there are two hyper-edges of weight 140 and 80. Assume that there are two datacenters (s_1 and s_2) that are directly connected, with d_1 and d_2 being hosted by s_1 and s_2, respectively. According to step (i), we first add the vertices s_1 and s_2. Then, by following the hyper-edge transformation procedure as stated earlier, the dashed hyper-edge is transformed into the set of vertices $\{d_1, d_{1'}, v_3, v_4\}$ with regular edges, while the solid one is transformed into the set of vertices $\{d_2, d_{2'}, v_1, v_2, v_3\}$ with regular edges. Even though v_4 belongs to both hyper-edges, it does not appear twice in the minimum cut graph. Next, we add in the graph the regular edges between VMs appearing in the application graph. Therefore, we add an edge between v_1 and v_2, as well as an edge between v_2 and v_3. The resultant minimum cut graph is shown in Fig. 8.

In the sequel, we apply the minimum cut maximum flow algorithm for undirected edges and find that the minimum cut is achieved by removing the edge between $d_{2'}$ and d_2, with the cost of cut being 80 MB. Specifically, we have chosen the augmenting path $\{s_1, d_1, d_1', v_3, d_2, d_{2'}, s_2\}$ to increase the flow by 80 MB, resulting in the maximum flow. According to the above augmenting path, the final residual graph is shown in Fig. 9. From the final residual graph, we observe that: **(a)** the set of VM vertices that are reachable from s_1 is $\{v_1, v_2, v_3, v_4\}$; and **(b)** the edge between d_2 and

$d_{2'}$ belong to the minimum cut. Consequently, all of the VMs are assigned on s_1, with d_2 being replicated on s_1. By performing the aforementioned assignment of VMs and data replicas, we result in a total network overhead of 80 MB.

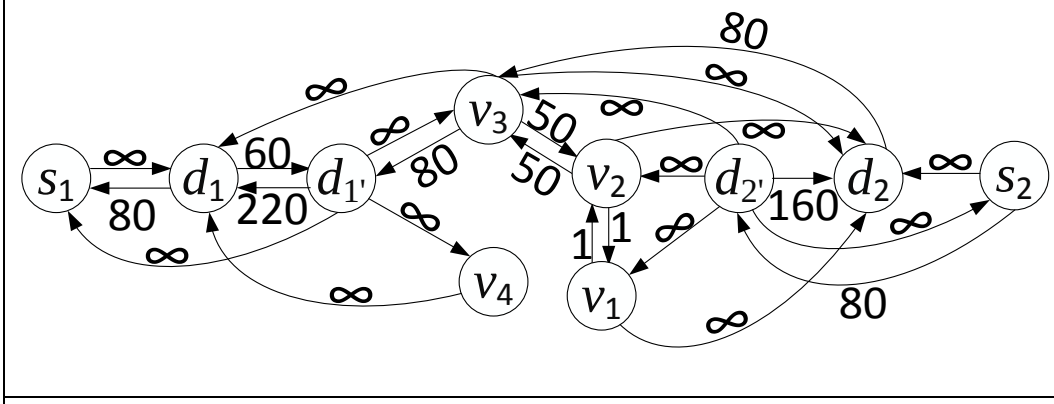

Figure 9. Final residual graph

3.3 HPA Functionality When Data Objects are Variously Accessed by VMs

In this section, we capture beyond the case where data objects are fully accessed by VMs, the following cases: **(a)** data objects are partially accessed VMs; and **(b)** data object are accessed more than one times by VMs. Below we explain the functionality of the algorithm when the aforementioned cases take place.

The only change against Section 3.2 when the above cases take place is the following: for any hyper-edge involving a data object d_k and m VMs, the weight of the edge for each of the m VMs (v_m) towards d_k, as well as the weight of the edge from $d_{k'}$ towards v_m, is equal to the volume of data needed for the access of d_k from v_m instead of infinite weight.

3.4 Algorithm Correctness and Optimality Issues

It must be noted that the following proofs hold under the assumption that no storage and computing capacity issues are taken into account. Note also that the below analysis holds also for star networks since such kind of networks can be considered as a special case of trees. It is also possible to apply modifications to the algorithm to behave also optimally for grid networks. However, such a discussion is out of the scope of this thesis.

Theorem 2. *Proof of correctness for the functionality of HPA as described in Section 3.2 when data objects are only fully accessed by VMs.*

Proof. In Theorem 1 we proved that the problem of two-datacenter case is reduced to the hyper-graph bi-partitioning problem. In [31] the hyper-graph bi-partitioning problem is reduced to minimum-cut problem by performing a hyper-edge transformation as shown in Section 3.2.1. ■

Theorem 3. *Proof of correctness for the functionality of HPA as described in Section 3.3 when data objects are only fully accessed by VMs.*

Proof. For any hyper-edge consisting of m VMs and a data object (d_k), with d_k being hosted by the source datacenter it holds one of the following results after applying the minimum cut on the graph: **(a)** all of the m VMs will be hosted by the source datacenter without d_k being replicated onto the terminal datacenter; **(b)** all of the m VMs will be hosted by the terminal datacenter along with the replication of d_k onto the terminal datacenter; **(c)** some of the VMs will be hosted by the source datacenter and the rest ones by the terminal datacenter, without replicating d_k onto the terminal datacenter; and **(d)** the same as (c), with the difference being that d_k is replicated onto the terminal datacenter.

For the weights of the edges between the m VMs and the vertices d_k and $d_{k'}$, we differentiate between the following cases: **(i)** the edges from the vertex $d_{k'}$ towards m VMs as well as the edges from VMs towards d_k are of infinite weight; and **(ii)** each edge from the vertex $d_{k'}$ towards one of the m VMs (v_m), as well as the corresponding

edge from v_m towards d_k has weight equal to the amount of data that v_m needs to access from d_k. Particularly, case (ii) reflects the functionality of HPA when VMs partially access data objects.

Consider that (a) is an optimal decision, then both (i) and (ii) result in that decision. Specifically, in both cases the cut take place at edges outside of the hyper-edge involving d_k and the m VMs. When (b) is an optimal decision, then both (i) and (ii) result in that decision, since in both cases the minimum cut contains the edge between d_k and $d_{k'}$. Note that (c) cannot be an optimal decision, since whenever some of the m VMs are to be hosted by the source datacenter and the rest ones by the terminal datacenter, the minimum-cut solution contains the replication of d_k onto the terminal datacenter. When (d) is an optimal solution, both (i) and (ii) result in that decision, with the edge between d_k and $d_{k'}$ belonging to the minimum cut. For the above to take place it must also hold that within the residual minimum-cut graph there are directed not disconnected paths from the source datacenter towards the VMs to be hosted by it. The aforementioned paths must cross at least one directed edge from d_q to $d_{q'}$, where d_q is a data object hosted by the source datacenter.

Therefore the changes described in Section 3.3 result always in the correct decisions. ■

Theorem 4. *Proof of correctness for the functionality of HPA as described in Section 3.3 when data objects are variously accessed by VMs.*

Proof. The proof is in the same line as that of Theorem 3. Specifically, the first two paragraphs of Theorem 3 hold exactly as is in this theorem.

Initially we make the assumption that the total amount of data being accessed from m VMs is strictly less than the size of d_k (as a result all of the m VMs partially access d_k). Consider that (a) is an optimal decision, then both (i) and (ii) result in that decision. Specifically, in both cases the cut take place at edges outside of the hyper-edge involving d_k and the m VMs. Note that (b) cannot be an optimal decision, since the optimal solution does not involve the replication of d_k onto the terminal datacenter. Note that only (ii) result in a decision without replicating d_k, with the minimum cut

containing the edges from $d_{k'}$ towards the m VMs. On the other hand, case (i) involves the replication of d_k onto the terminal datacenter since the edges from $d_{k'}$ towards the m VMs cannot be cut because their weight is infinite. When (c) is an optimal decision, only case (ii) can result in such a decision. The above takes place when cutting the edges from $d_{k'}$ towards the VMs to be hosted by the terminal datacenter. Note that the above cannot happen in case (i) because the edges from $d_{k'}$ towards the VMs to be hosted by the terminal datacenter are of infinite weight and thus cannot be cut. Again (d) cannot be an optimal solution, because the replication of d_k onto the terminal datacenter is suboptimal.

We make the assumption that the total amount of data being accessed from m VMs is strictly greater than the size of d_k, with some VMs partially accessing d_k, some other fully accessing d_k, while the rest ones accessing d_k more than one time. If (a) is an optimal decision, then both (i) and (ii) result in such a decision (as explained previously). When (b) is an optimal decision, then both (i) and (ii) result in that decision. Specifically, both cases (i) and (ii) result in a minimum cut containing the edge between d_k and $d_{k'}$. Regarding (c), if it is an optimal decision, then both (i) and (ii) result in that decision, under the prerequisite that the total amount of data accessed by the VMs to be hosted by the terminal datacenter is greater than or equal to the size of d_k (we do not consider the case where the data accessed by the aforementioned VMs is less than the size of d_k because such a case is described in previous paragraph). When (d) is an optimal decision, then both (i) and (ii) result in that decision (see the corresponding explanation of the previous paragraph).

As a result the changes described in Section 3.3 lead always in the correct decisions. ■

Corollary 1. *From Theorem 4 we observe that the problem of the two-datacenter case when data objects are variously accessed by VMs can be correctly found by solving the minimum {s-t} cut problem*.

Theorem 5. *HPA is optimal for the two-datacenter case when data objects are fully accessed or variously accessed by VMs*

Proof. In Theorem 1 it is proved that when the underlying network consists of two datacenters, then our problem (considering that data objects are only fully accessed by VMs) is reduced to the hyper-graph bi-partitioning problem. According to [31] the problem of hyper-graph bi-partitioning can be solved optimally by reducing it to the problem of finding a cut of minimum capacity. Therefore by applying hyper-graph partitioning in a network of two datacenters the problem is solved optimally when data objects are fully accessed by VMs.

What remains to prove is that HPA is also optimal when data objects are variously accessed by VMs. From Section 3.3 we observe that the problem is still solved by finding the minimum cut between the source and terminal datacenters. Corollary 1 states that HPA solves the two-datacenter case by solving the minimum {s-t} cut problem. Because the minimum {s-t} cut always results in the optimal solution [3], HPA results also in the optimal solution.

1:	mark all of the datacenters as unexplored
2:	choose randomly a datacenter and mark it as explored
3:	assign all of the VMs on the aforementioned datacenter
4:	**for each** unexplored datacenter s_i that is 1-hop away from any explored datacenter s_j
	mark s_i as explored
5:	draw a new min-cut graph and add s_i and s_j
6:	**for each** data object d_k belonging to D
7:	identify a hyper-edge/regular edge e for d_k considering only VMs assigned on s_j
8:	**if** e contains only one VM
9:	draw a regular edge of weight $\delta(d_k)$ between the respective VM and s_i/s_j
10:	**else**
11:	transform the hyper-edge into a set of regular edges as explained in §3.2.1 and §3.3
12:	declare u_x as d_k, and u_y as $d_{k'}$
13:	add a directed edge of infinite weight from s_i/s_j towards d_k
14:	add a directed edge of infinite weight from $d_{k'}$ towards s_i/s_j
15:	**end if**
16:	**end for**
17:	**for each** edge e between a VM v_g hosted by an explored datacenter besides s_i and s_j and a VM v_f currently assigned on s_j
18:	draw an edge between v_f and s_j with the same weight as that of e
19:	**end for**
20:	apply max-flow min-cut algorithm to re-assign VMs and mark the replication of data objects onto datacenters according to §3.2.2
21:	**end for**
22:	The replication of data objects take place in a store-and-forward way, starting from the primary replicas. If there has been a decision for replicating a data object d_k at a datacenter s_i, but there is no VM that accesses the replica of d_k onto s_i, then revoke the respective replication.

Figure 10. Pseudocode of HPA

4 Extending HPA to Consider a Tree Structured Network of Multiple Datacenters

In this section we extend the aforementioned algorithm for the problem of two datacenters to a more general algorithm for tree-structured network of multiple datacenters. Note that HPA solves optimally our problem for a tree-structured network of multiple datacenters; however due to limited space we omit the proofs.

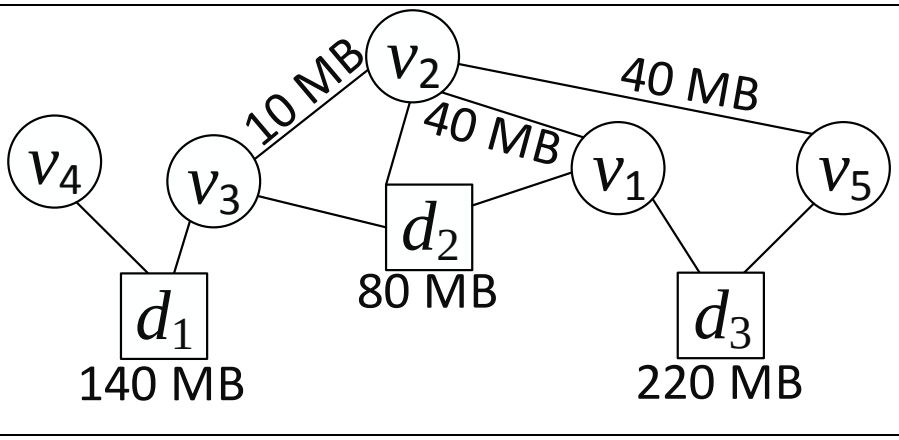

Figure 11. Application

The pseudocode of the extended algorithm (called HPA) is shown in Fig. 10. Below we give an explanation of the pseudocode: **(a)** All of the datacenters are marked as unexplored (line 1). In the sequel, a datacenter is randomly chosen and marked as explored (line 2). All of the VMs are temporarily assigned on the aforementioned datacenter (line 3). **(b)** For each unexplored datacenter choose one (s_i) that is 1-hop away from any explored datacenter s_j (line 4); **(c)** s_i is marked as explored and a new minimum cut graph is drawn by adding s_i and s_j (line 5-6). **(d)** For each data object d_k, identify the VMs hosted by s_j and access d_k (line 7). In case of exactly one VM, draw a regular edge between the respective VM and either s_i if d_k's hosting datacenter is

closer to s_i, or otherwise s_j (line 8-11). In case of more than one VM, identify the respective hyper-edge and transform it into a set of regular edges as explained in Section 3.2.1 and Section 3.3; add a directed edge of infinite weight originating from either s_i provided that s_i is closer to the d_k's hosting datacenter against s_j, or otherwise s_j, and ending on d_k; add also an edge of infinite weight originating from $d_{k'}$ and ending on s_i, provided that s_i is closer to the d_k's hosting datacenter against s_j, otherwise on s_j (line 12-15). **(e)** For any edge e between a VM v_g hosted by the explored datacenters besides s_i and s_j and a VM v_f currently assigned on s_j, we draw an edge between v_f and s_j with the same weight as that of e. The above is because v_g is guaranteed to be closer to s_j against s_i; since currently only the explored datacenters have been assigned VMs, and these datacenters are closer to s_j against s_i (line 18-20). **(f)** Apply the maximum flow minimum cut algorithm to temporarily re-assign the VMs and mark the replication of data objects onto datacenters as explained in Section 3.2.2 (line 21). Note that the replication is just marked but not performed at the current step. **(g)** The last step revolves around the way a data object (d_k) is transferred from the datacenter hosting the primary replica of d_k towards the datacenters marked by the algorithm to host the replicas of d_k in a store-and-forward way (line 23). **(h)** This step concerns the revocation of a replication in case a decision has been made for replicating a data object d_k at a datacenter s_i, provided that there is no VM that accesses the replica of d_k onto s_i (line 24).

4.1 HPA Functionality

To illustrate the functionality of HPA, we set forth the following example. Consider an application, shown if Fig. 11,consisting of five VMs accessing three data objects (d_1, d_2, and d_3).The network is consisted of three datacenters s_1, s_2, and s_3 hosting d_1, d_2, and d_3, respectively.

The algorithm begins by marking all of the datacenters as unexplored. In the sequel, it marks randomly s_1 as explored and assigns all of the VMs onto s_1. Because s_1 has s_2 as the only one 1-hop neighbor, s_2 is chosen and marked as explored. Initially, the minimum cut graph is empty, with s_1 and s_2 being added on it. Three hyper-edges are identified regarding the data objects d_1, d_2, and d_3 (see Fig. 12).

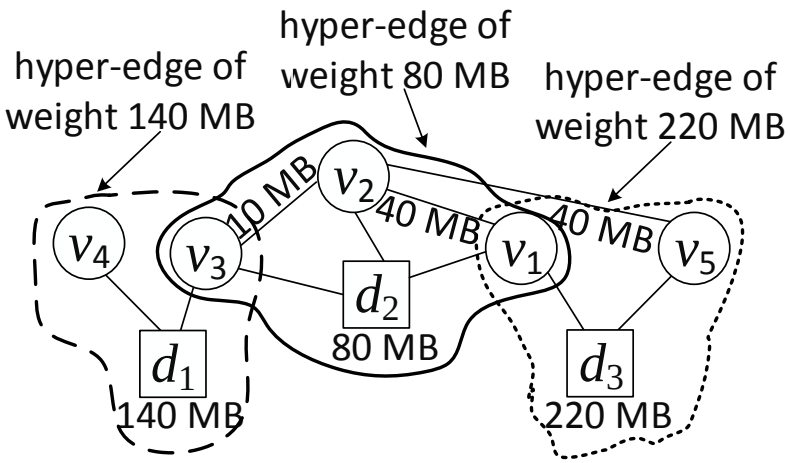

Figure 12. Hyper-edges identified when considering s_1 and s_2

The transformation of each hyper-edge into a set of regular edges takes place according to Section 3.2.1 and Section 3.3. Because the hosting datacenter of the primary replica of d_1 and d_2 is s_1 and s_2, respectively; an edge of infinite weight is drawn from s_1 towards d_1 (as well as an edge of infinite weight from $d_{1'}$ towards s_1) and another one from s_2 towards d_2 (as well as an edge of infinite weight from $d_{2'}$ towards s_2). On the other extreme, the hosting datacenter of the primary replica of d_3 is closer to s_2 than s_1; thus an edge of infinite weight is drawn from s_2 towards d_3, as well as an edge of infinite weight from $d_{3'}$ towards s_2. Since all of the VMs have been temporarily assigned on s_1, the lines 18-20 are not executed. The minimum cut graph that has been created is shown in Fig. 13.

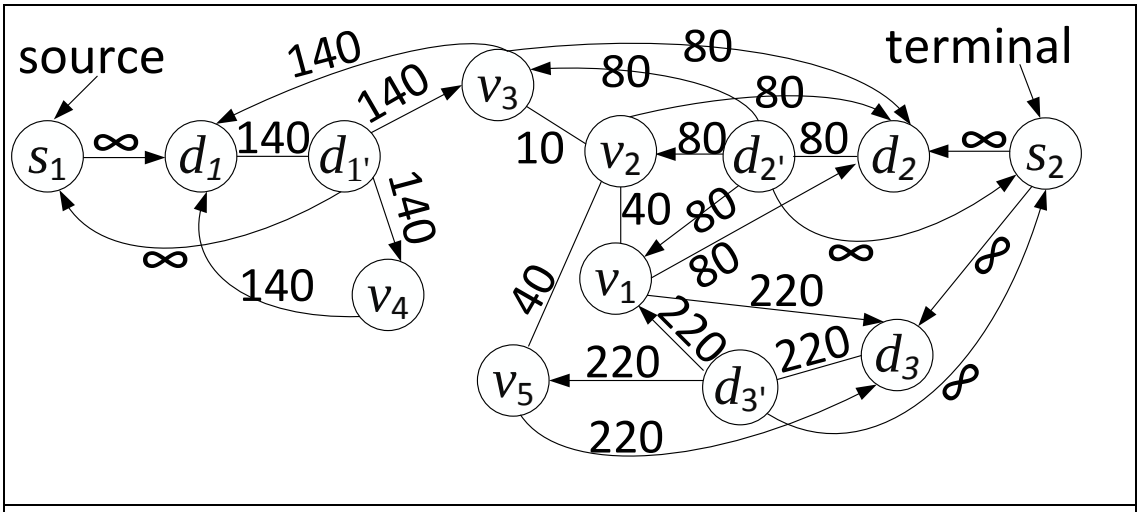

Figure 13. Min-cut graph when considering s_1 and s_2

After executing the maximum flow minimum cut algorithm regarding the graph of Fig. 13, we result in the residual graph shown in Fig. 14. It is observed that the minimum cut equals 90 (80+10), with the reachable set of VMs from s_1 being {v_3, v_4}. Therefore, v_3 and v_4 are assigned onto s_1, while d_2 is replicated onto s_1 since the edge between d_2 and $d_{2'}$ belong to the minimum cut. On the other extreme, v_1, v_2, and v_5 are assigned onto s_2. It must be noted that the assignment/replication is temporary because s_3 has not been explored yet.

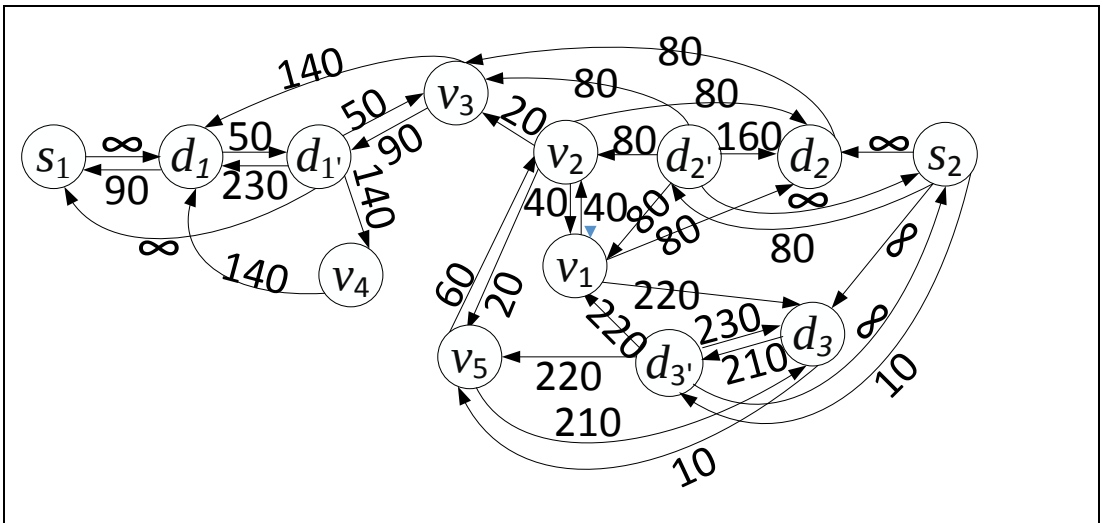

Figure 14. Residual graph when considering s_1 and s_2

In the sequel, s_3 is the next (and last) datacenter that is marked unexplored, which is 1-hop away from s_2. Considering the VMs hosted by s_2 (which plays the role of s_j in the pseudocode), two hyper-edges are identified shown in Fig. 15. The

transformation of each hyper-edge into a set of regular edges takes place according to Section 3.2.1 and Section 3.3. Because the hosting datacenter of the primary replica of d_2 is s_2, an infinite edge is drawn from s_2 towards d_2, as well as an edge of infinite weight from $d_{2'}$ towards s_2. On the other extreme, the hosting datacenter of the primary replica of d_3 is s_3; therefore an edge of infinite weight is drawn from s_3 towards d_3, as well as another one from $d_{3'}$ towards s_3. Due to the fact that there exists an edge between v_2 (hosted by s_2) and v_3 (hosted by s_1), an edge is drawn between v_2 and s_2. The reason we choose s_2 and not s_3 as the one end of the edge is that the hosting datacenter of v_4 is closer to s_2 than s_3. The aforementioned are according to the lines 18-20 of HPA's pseudocode. The minimum cut graph is depicted in Fig. 16.

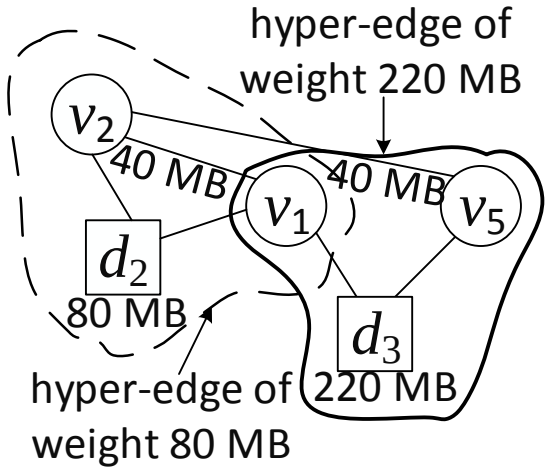

Figure 15. Hyper-edges identified when considering s_2 and s_3

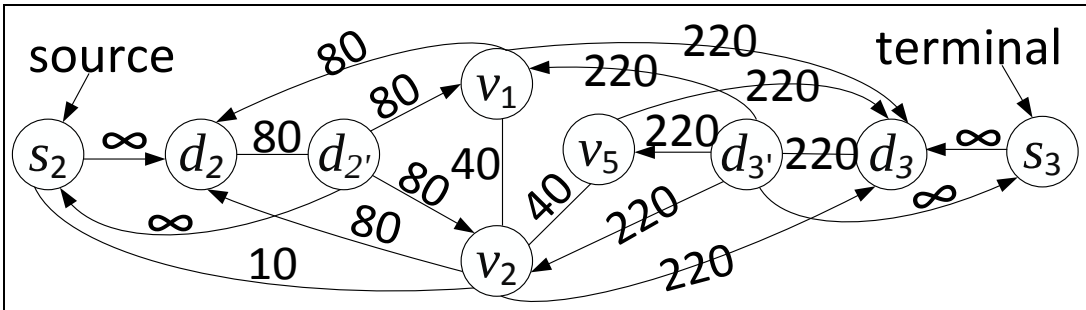

Figure 16. Min-cut graph when considering s_2 and s_3

By applying the minimum cut maximum flow algorithm for the graph shown in Fig. 16, we result in the residual graph depicted in Fig. 17. As can be seen, the minimum cut equals 90 (80+10), with the reachable set of VMs from s_3 being $\{v_1, v_2, v_5\}$. Consequently, the set of VMs $\{v_1, v_2, v_5\}$ is assigned onto s_3. Because the edge between d_2 and $d_{2'}$ is being cut, d_2 is replicated at s_3. The data object d_2 is first transferred towards s_2 (from s_1) and then towards s_3 from s_2 (line 23). Note that the replica of d_2 onto s_2 is not accessed by any VM within the network. Therefore, the replication of d_2 onto s_2 is revoked (line 24). The final VM assignments and data replications are shown in Fig. 18, with the total network overhead being 180 MB.

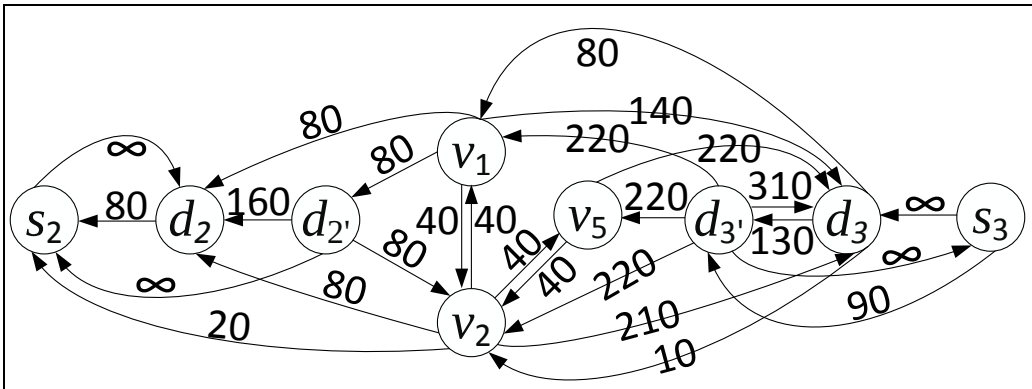

Figure 17. Residual graph when considering s_2 and s_3

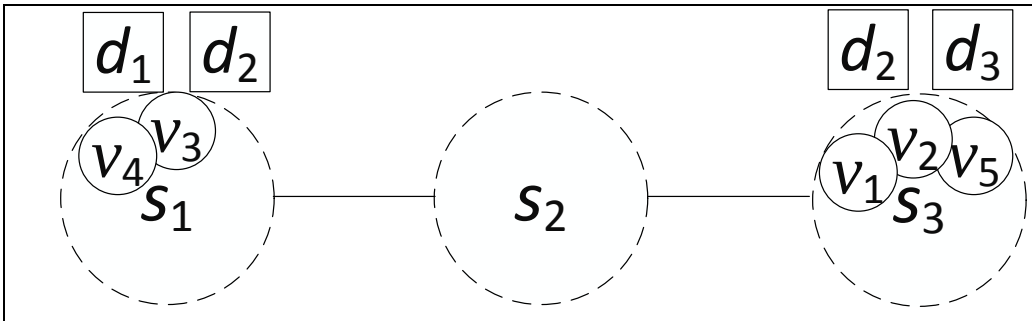

Figure 20. Final VM assignment and data replication

4.2 Optimality Issues

In this section, we prove the optimality of HPA when the network is structured as a

tree. We also assume that no storage and computing capacity issues are taken into account.

Theorem 6. *HPA results always in the optimal solution when the network is structured as a tree.*

Proof. First we must prove that the first time the minimum cut is solved for a pair of datacenters (s_i, s_j) then the solution of VM assignment reflects that if a VM (v_m) is assigned onto the one of the two datacenters taking place in the minimum cut (let s_i), then the optimal solution is to assign v_m onto either s_i or any datacenter that is closer to s_i against s_j. Note that a data object is connected to either s_i, if its primary replica is hosted by either **(a)** s_i or any datacenter that is closer to s_i other than s_j; or **(b)** s_j if its primary replica is hosted by either s_j or any datacenter that is closer to s_j other than s_i. Therefore, s_i represents itself as well as all of the datacenters that are closer to s_i against s_j; while s_j represents s_j as well as all of the datacenters that are closer to s_j against s_i. Now to result in contradiction consider that the minimum cut solution assigns v_m onto s_i, while there is a solution with less network overhead against that of minimum cut, which assigns v_m onto any datacenter that is closer to s_j other than s_i. If that solution assigns v_m onto s_j, then we result in contradiction since minimum-cut results in the optimal solution when having to choose between s_i and s_j. If the aforementioned solution assigns v_m onto any datacenter that is closer to s_j against s_i, then we also result in contradiction. The above is due to the following. By the minimum-cut we know that v_m has greater dependence on the set of VMs and data objects belonging to s_i against the set of VMs and data objects belonging to s_j. Therefore, by assigning v_m onto a datacenter that is closer to s_j other than s_i incurs greater network overhead against assigning v_m onto s_i (contradiction).

Secondly, we must prove that when solving the minimum-cut for a pair of datacenters (s_i, s_j) and the solution contains the replication of a data object d_k onto one of the datacenters (let s_j), then the optimal solution contains the transfer of d_k from s_i towards s_j even if finally s_j does not host a replica of d_k. To result in contradiction, consider that the minimum-cut solution contains the replication of d_k onto s_j and finally d_k is not transferred from s_i onto s_j. In order for d_k to not cross the edge between s_i and s_j, it must hold the following. The total amount of data, regarding d_k,

accessed by the VMs assigned onto s_j must strictly less than the size of d_k. However, the above comes in contradiction with the functionality of HPA when it is applied between two datacenters. Specifically, the functionality of HPA states that if d_k is decided to be replicated onto s_j, then the total amount of data, in terms of d_k accessed by the VMs (currently) assigned on s_j is greater than or equal to the size of d_k. Now consider the case where d_k is decided to be replicated at s_j but later it is revoked since no VM accesses the replica of d_k at s_j. Even in the case that the aforementioned VMs that are temporarily hosted by s_j and finally are not hosted by s_j, it must hold the following. According to the previous paragraph, those VMs must be hosted by any datacenter that is closer to s_j against si. Therefore, there must be a replication of d_k to at least of one datacenters that is closer to s_j against s_i.

Third, we prove that when the VMs are temporarily assigned to some datacenter (s_y) and the minimum cut is applied on a pair of datacenters (s_i, s_j) the following must happen to not result in sub-optimal solutions. If there is an edge between a VM (v_f) participating in the minimum cut and a VM (v_g) temporarily assigned on s_y, then an edge must be drawn between v_f and the datacenter that is closer to s_y among s_i and s_j (let s_j). The weight of that edge is the same as that between v_f and v_g. Consider now that we do not draw that edge, and its weight is substantially big. Then the minimum cut may falsely result in the assignment of v_f onto s_i; because as shown in first paragraph, v_f moves away from s_y when it is assigned on s_i against s_j. On the other hand, if the edge is drawn on s_i instead of s_j, then again the minimum cut may falsely result in the assignment of v_f onto s_i; because as shown in first paragraph, v_f moves away from s_y when it is assigned on s_i against s_j.

According to the above, when exploring all of the datacenters, that is, applying the minimum cut between all of the pairs of one-hop datacenters, we result in the optimal assignment of VMs as well as the replication of data objects onto datacenters. However, this assignment does not guarantee the optimal solution in terms of network overhead, since it does not provide the way the data objects are replicated onto datacenters.

At step (h) in Section 4, we explain that the data objects are replicated in a store-and-forward way starting from the datacenters hosting the primary replicas. In that

way, a datacenter (s_i) fetches a data object d_k from the nearest datacenter (s_j) located between s_i and the datacenter hosting the primary replica of d_k. It holds that: **(a)** there is no transfer of d_k that cross an edge more than one time due to the store and forward way; **(b)** there is a single path between the datacenter hosting the primary replica of d_k and a datacenter that is to host a replica of d_k (due to the tree topology). Therefore, according to (a) and (b) the replication of data objects take place in an optimal way. ■

Note that we can run HPA independently for each independent graph of VMs and data objects within the system without any compromise in the quality of the overall solution. It must be also noted that the algorithm can be easily adapted to tackle the case of new additions of VMs and data objects within the system as follows: **(i)** the new VMs and objects introduced within the system form an independent graph against the rest VMs and data objects within the system. In such a case, the algorithm is executed only for the new graph without affecting the overall solution. **(ii)** When the new VMs/data objects have dependencies with the rest VMs and data objects within the system, then we update all the independent graphs within the system and we re-execute the algorithm only for the independent graphs that have been affected by the new VMs and data objects. Unfortunately, in such a case there are no guarantees for optimality due to the existing data object replicas within the system.

5 Hypergraph Partitioning Algorithm when Considering Storage and Computing Capacity Constraints

This section discusses the extension of the algorithm proposed in Section 3 to consider storage and capacity constraints.

5.1 Extending HPA for the Two Datacenter Case when Considering Capacity Constraints

In this section, the VM assignment and data replication problem is solved for a system consisting of two datacenters (s_1, s_2). Initially, the problem is solved in the same way as that described in Section 3.2.2. After resulting in the solution for the un-capacitated case, we perform the following steps:

Step 1. If there is no storage and computing capacity violation in any datacenter within the system, then the replication takes place as dictated by the solution for the un-capacitated case. In case of both storage and computing capacity violation, we proceed to step 2. In case of only computing capacity violation we proceed to step 3.

Step 2. We replicate in an iterative fashion the data object that its replication reduces the network overhead as much as possible. The data objects that have been decided to be replicated, but the replication did not take place due to the storage capacity constraints are marked as "non-replicable". Each hyper-edge involving a non-replicable data object is transformed into a set of regular edges as follows. For each VM participating in such a hyper-edge we add a normal edge between the respective VM and the datacenter hosting the corresponding data object. The weight of such an added edge equals the amount of data, regarding the corresponding data object,

accessed by the respective VM. A new assignment is obtained by applying the minimum-cut on the resultant graph containing only regular edges. Next, we investigate whether any datacenter's computing capacity is violated when performing the new assignment. When no violation takes place, the algorithm terminates and the solution of this step is considered as the final solution. In case of violation, step 3 takes place.

Step 3. . In this step, we assume that the computing capacity of the one datacenter (let s_1) is violated under the VM assignment obtained in step 2. Note that there cannot be a case where the computing capacity of both datacenters is violated. The above is because of the assumption that there is at least one VM assignment that does not violate the computing capacity of both datacenters. For each VM hosted by s_1 (according to the assignment obtained from step 2), we calculate which is the impact in the network overhead if the corresponding VM is re-assigned onto s_2. (The calculation of the impact is described in next paragraph). The VM re-assignment that burdens the system with the minimum network overhead is decided to be performed. The aforementioned re-assignment process iterates itself until there is a feasible assignment. In case of no feasible assignment, the algorithm terminates with no assignment.

5.2 Extending HPA for the Tree Structured Network of Multiple Datacenters

In this section, the VM assignment and data replication problem is solved for a system consisting of N datacenters structured as a tree. The procedure is identical with that of Section 5.1 up to step 2. Regarding step 3 the following take place. For each datacenter that its computing capacity is violated, we attempt to find a re-assignment of its VMs towards other datacenters such that its computing capacity is not violated. Among all of the feasible hosts for the re-assignment of a VM, we choose the one that burdens the system with the least network overhead. The above process is iterated until there is no computing capacity violation or there is no feasible VM re-assignment.

VM re-assignment impact. The impact of re-assigning a VM v_i from s_x to s_y is the same as that described in Section 5.1, with the differences being that here **(a)** there are also other datacenters within the system other than s_x and s_y, and **(b)** the distance between s_x and s_y may be more than one.

6 Evaluation

6.1 Setup

The experimental evaluation has been conducted on NS-2 [36]. Twenty different network topologies were generated through the internet topology generator Inet-3.0 [30], with the number of datacenters being uniformly chosen between 10 and 50. The corresponding tree-based routing topology is obtained by constructing a spanning tree, whereby each pair of datacenters is connected via a single path. The total number of VMs and data objects within the system is around 20000 and 100000, respectively. The aforementioned settings are chosen such that 200 independent graphs of VMs and objects exist within the system. The maximum independent graph consists of 200 VMs and 1000 objects, while the minimum one consists of 20 VMs and 100 objects. Note that we do not consider larger independent graphs (even though our algorithm can support larger settings) due to the inherent properties of mobile cloud applications (small sizes). It must be noted that within an independent graph more than one application may exist. Because data objects sizes follow the lognormal distribution [5], data volumes ranged approximately between 10 MB to 1 GB by following the lognormal distribution with parameter settings $\mu = 0$ and $\sigma = 1$. A VM can access a data object approximately between 0.1 and 10 times (lognormal distribution with the same parameters as previously).

The behavior of HPA is compared with state-of-the-art algorithms from the literature. Specifically, the Distributed Bartering Algorithm (DBA) [19] performs VM assignment without taking into account group VM Migrations. On the other hand, the Distributed Re-assignment Algorithm (DRA) [25] performs not only single but also group VM Migrations. Unfortunately, the aforementioned algorithms do not consider

at all the replication of data objects accessed by VMs. For this reason, we include in our comparisons an algorithm that performs replication of data objects. Among the heuristics employed in the literature for data object replication, we choose a greedy (called Greedy) [2] heuristic which is widely known in the literature.

6.2 Capacity-Free Datacenters

The first set of experiments was conducted without considering computing and storage capacity constraints on datacenters. It must be noted that HPA solves the initial VM placement problem, while DBA and DRA decide the transition from an old VM assignment scheme towards a new one. Therefore, for comparison reasons, we first assigned randomly the VMs onto datacenters and then we applied DBA and DRA on top of the random VM assignment. On the other extreme, we executed HPA without needing an initial random VM assignment. It must be noted that the results were normalized according to the algorithm yielded the worst performance (i.e., DBA). As observed in Fig. 19, where the communication between VMs is high, HPA achieves superior performance against DRA, Greedy, and DBA. Specifically, HPA achieves a network overhead reduction of 53%, 47%, and 32% against DBA, Greedy, and DRA, respectively. The reason that DBA results in the worst performance is that for the decision of the VM migrations from an old assignment scheme to the new one, it considers only single VM migrations. DRA takes into consideration the migration of VMs in a grouped manner resulting in better results against DBA. On the other extreme, Greedy cannot achieve a good performance due to the fact that it is not capable of migrating VMs at all. The superiority of HPA is attributed to the following fact. HPA considers replicating data objects, with the VMs having the option to access those data objects from any datacenter holding their replicas. In that way, the network overhead can be significantly reduced since a datacenter can access a data object from the nearest datacenter holding the replica of that data object instead of accessing it from the datacenter holding the primary replica of the respective object.

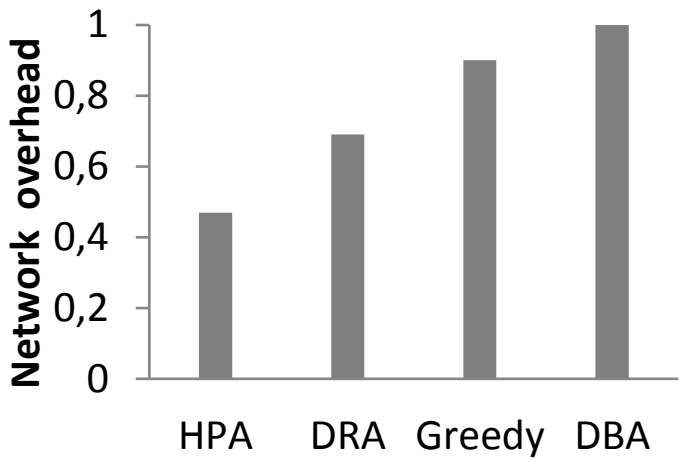

Figure 19. Network overhead with high communication between VMs

In Fig. 20, we can see that HPA achieves a slightly better performance against Greedy. This is reasonable, since the communication between VMs is low; thus what mostly counts is the replication of data objects. On the other extreme, DRA and DBA achieve by far the worst performance against HPA and Greedy. The above is justified by the fact that DRA and DBA do not consider at all the replication of data objects.

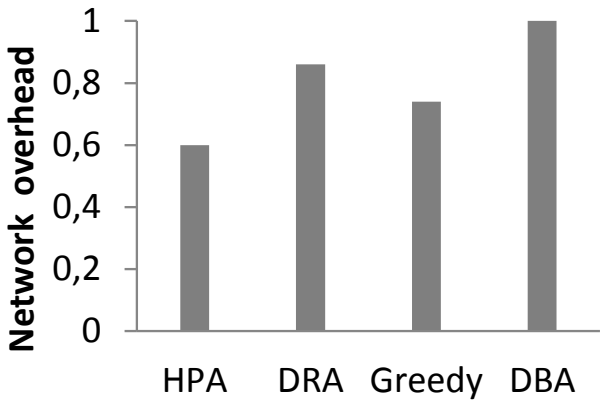

Figure 20. Network overhead with low communication between VMs

In Table I, we show the execution times of the most competitive algorithms, i.e., HPA and DRA. We show only the execution times for the case of single independent graphs, since for multiple independent graphs we can simply aggregate the execution times. As we can see, even though the time complexity of HPA increases when

increasing the number of VMs and data objects, its execution time is acceptable even for large settings.

Table I. Execution Times

Algorithms / settings	(VMs, Objects) (20,100)	(VMs, Objects) (100,500)	(VMs, Objects) (200,1000)
HPA (secs)	0.06	1.34	3.75
DRA (secs)	0.005	0.07	0.25

6.3 Capacitated Datacenters

In this section, we conducted experiments for the comparison of HPA with DRA, Greedy, and DBA under computing and storage capacity constraints. Initially, the computing capacity of each datacenter within the system was fixed to the amount of the total computing capacity requirements needed to host the VMs under a random VM initial assignment. The results were normalized according to the algorithm yielded the worst performance (i.e., DBA). In the first set of experiments, we varied the surplus computing capacity of each datacenter within the system from 10% to 50%, with the communication between VMs being kept high. As we can see from Fig. 21, HPA achieves the best performance even when the surplus computing capacity of datacenter is tight. Specifically, when the surplus computing capacity of each datacenter equaled 10%, HPA yielded a network overhead reduction of roughly 5% and 2% against DBA and DRA, respectively. On the other extreme, when the surplus computing capacity of each datacenter was relaxed, the HPA achieved a bigger network overhead reduction against DBA and DRA. Specifically, when the surplus computing capacity of each datacenter was equal to 50, the network overhead reduction of HPA compared to DBA and DRA was 48% and 15%, respectively. We must note that we do not show at all the performance of Greedy, because it does not consider at all VM migrations and thus computing capacity.

As we can see in Fig. 22, HPA overcomes DRA and DBA. Specifically, we observe that as the surplus computing capacity increases, the performance of HPA increases in a clearly higher rate against that of DRA. This is because, replication is quite significant when the communication of VMs is low. Again, the performance of Greedy is not shown for the same reason as previously.

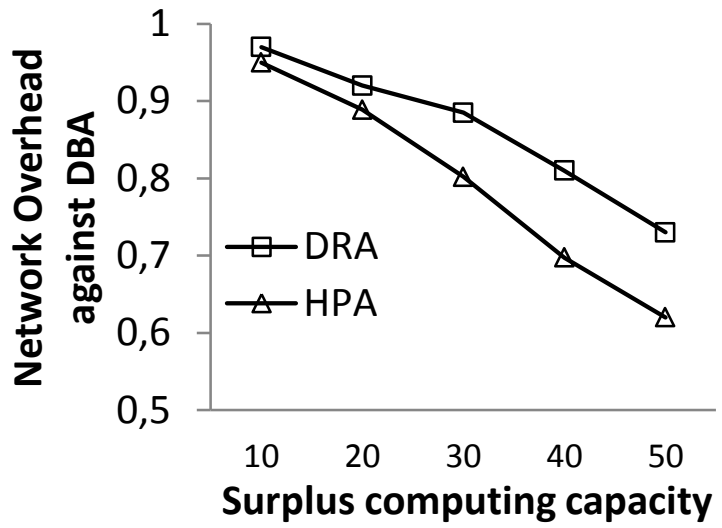

Figure 21. Network overhead with high communication between VMs, when varying surplus computing capacity.

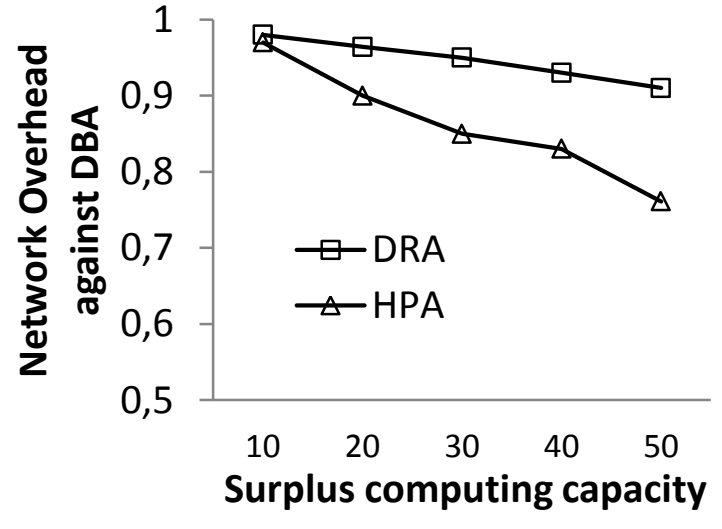

Figure 22. Network overhead with low communication between VMs, when varying surplus computing capacity

The second set of experiments was conducted to investigate the behavior of HPA when varying the storage capacity. Initially, the storage capacity of each datacenter within the system was fixed to the amount of the total storage capacity requirements needed for hosting within the system the initial replica of all of the data objects. We varied the surplus storage capacity of each datacenter from 20% to 100%, with each of them having size equal to the average data object size. In Fig. 23 we show the

performance of HPA, and Greedy against DBA when the communication between VMs is high. The reason that we do not show DRA is that it does not consider the storage capacity of datacenters. As can be seen, HPA is superior against both Greedy and DBA. This is because of the capability of HPA to handle both data object replication and VM Migration. On the other hand, Greedy has the worst performance in most cases. This is because, the communication of VMs is high and Greedy is not able to perform VM migrations, while in many cases its capability of to perform data object replication is limited due to storage capacity limitations. Regarding DBA, it is able to perform VM migrations to mitigate network overhead.

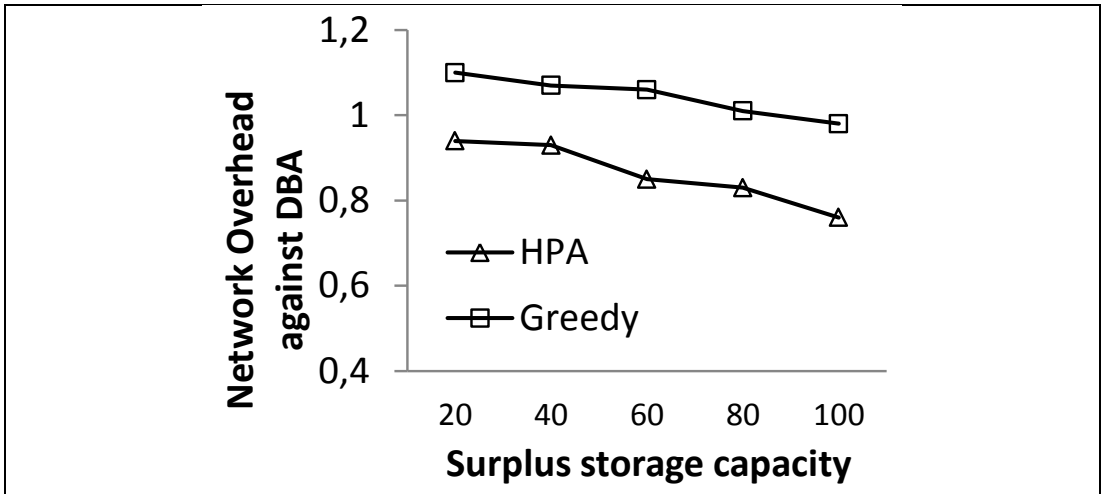

Figure 23. Network overhead with high communication between VMs, when varying storage capacity.

The same experiment is also conducted when the communication of VMs is low. As seen, Greedy is superior against HPA in all cases. The above is because the benefit of migrating VMs is limited in this scenario due to the low communication between VMs.

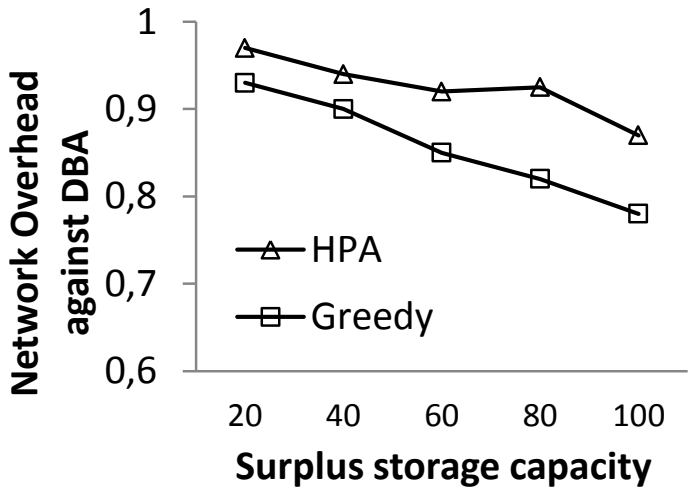

Figure 24. Network overhead with low communication between VMs, when varying storage capacity.

7 Related work

The replica placement problem has been researched quite extensively, and a variety of problems definitions have been proposed [15]. A system called DepSky is presented in [4] to improve availability, integrity and confidentiality. Ref. [27] tackle the problem of minimizing the time needed for the transition from an old replica assignment scheme to a new one. The difference between the above problems and our problem lies in the fact that the objective function is not the same. A dynamic replication method is used in [8] to pre-replicate data based on predicted future needs. The above is different against our problem in the sense that data are replicated based on abstract needs that are not related with interdependent VMs. The problem of workload-aware data placement and replication is tackled in [11] by applying a hyper-graph partitioning technique. The main difference between this thesis and that of [11] is that the latter one considers: (a) transactions using data, with the partitioning taking place based on the frequencies that the respective transactions use data; and (b) a fully connected topology, which is the case for data-base servers but not the case for a network between datacenters. A widely known greedy algorithm for replicating data objects onto capacitated nodes is discussed in [2]. The problem of minimizing the transition cost from an old replica assignment scheme to a new one is tackled in [13] and [14]. On the other hand, authors of [22] and [23] tackle the aforementioned problem under the objective of minimizing the transition time from an old replica assignment scheme to a new one.

Our work is closely related with the virtual machine (VM) placement problem [6]. The problem of workload consolidation is tackled in [9] and [18] to minimize energy consumption. The VM placement problem is addressed in [7], whereby the objective is to minimize the network congestion within the system. A network aware VM placement approach is proposed in [29] to improve slowdown. The same problem is also tackled in [1], with the objective being to minimize the maximum access latency

between the communicating VMs. The dynamic service placement problem is tackled in [32], with the objective being to reduce the hosting cost over time according to both demand and resource price fluctuation. In [16] and [28], the authors target the VM placement problem with their objective being the same with that of our problem. However, the main difference between [16] and our problem is that the former does not consider dependencies between VMs and data. A fully distributed algorithm is proposed in [19], called DBA, to solve the same problem tackled in this paper under the context of clouds. DBA works for general-structured graphs and takes into account capacity constraints on nodes. The difference with our approach is that DBA does not consider migrating group of VMs, resulting in that way in sub-optimal placements. On the other hand, DRA [25], [24] is an optimal fully distributed algorithm working also for general-structured application graphs and taking into consideration capacity constraints on nodes.

The data movement plan is addressed in [12] in the context of query processing to minimize network overhead. The problem is based on hyper-graph partitioning for tree-structured networks. The main difference between [12] and our work is the system model. Specifically, [12] decides only the movement of data within the network, without examining replicas of data objects. It also assumes that data are only fully accessed by operators. The agent migration problem is tackled in [26] to minimize the network overhead between communicating agents. The difference between the above work and ours is that the former does not consider replication of data.

To the best of my knowledge, DBA [19], DRA [25], and [2] are the most closely connected works with this thesis.

8 Conclusions and outlook

In this thesis, I have formulated the joint problem of replicating data objects and assigning the virtual machines onto datacenters. An algorithm is proposed to solve the aforementioned problem that is based on hyper-graph partitioning. An extension of the algorithm has also been designed to tackle the problem when considering storage and computing capacity constraints on datacenters. The proposed technique was compared to two state-of-the-art algorithms found in the literature, named DBA [19], DRA [25], and Greedy [2]. The experimental evaluation showed that HPA can achieve a network overhead reduction of up to 50% and 30% against DBA and DRA, respectively. Our future directions include addressing the problem for the transition of an old replica and VM assignment scheme to a new one.

References

[1] M. Alicherry, T. V. Lakshman, "Network Aware Resource Allocation in Distributed Clouds,"*IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[2] S. Bakiras, T. Loukopoulos, D. Papadias, I. Ahmad, "Adaptive Schemes for Distributed Web Caching," *Journal on Parallel and Distributed Processing (JPDC)*, 65(12), pp. 1483-1496, 2005.

[3] D. P. Bertsekas, "Network Optimization: Continuous and Discrete Models," *Belmont: Athena Scientific*, 1998.

[4] A. Bessani, M. Correia, B. Quaresma, F. Andre, P. Sousa, "DepSky: dependable and Secure Storage in a Cloud-of-clouds," *ACM Transactions on Storage*, 9(4), 2013.

[5] E. Chlebus, G. Divgi, "A Versatile Probability Distribution for Light and Heavy Tails of Web File Sizes," *Wireless Communication and Networking Conference (WCNC)*, 2009.

[6] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S.Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, S. U. Khan, and A. Y. Zomaya, "A Survey and Taxonomy on Energy Efficient Resource Allocation Techniques for Cloud Computing Systems," *Computing*, 2015.

[7] J. W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, "Joint VM Placement and Routing for Data Center Traffic Engineering," *IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[8] L. M. Khanli, A. Isazadeh, T. N. Shishavan, "PHFS: A Dynamic Replication Method to Decrease Access Latency in the Multi-tier Data Grid," *Future Generation Computing Systems*, 27(3), 233-244, 2011.

[9] M. Khelghadoust, V. Gramoli, D. Sun, "GLAP: Distributed Dynamic Workload Consolidation through Gossip-Based Learning," *IEEE International Conference on Cluster Computing*, 2016.

[10] T. Kosar, E. Arslan, B. Ross, B. Zhang, "StorkCloud: Data Transfer Scheduling and Optimization as a Service," *in 4thACM Workshop on Scientific Cloud Computing (ScienceCloud)*, New York, NY, USA, 29-36, 2013.

[11] K. A. Kumar, A. Quamar, A. Deshpande, S. Khuller, "SWORD: Workload-aware Data Placement and Replica Selection for Cloud Data Management Systems," *The VLDB Journal-The International Journal on Very Large Data Bases*, 23(6), 845-870, 2014.

[12] J. Li, A. Deshpande, S. Khuller, "Minimizing Communication Cost in Distributed Multi-query Processing," *International Conference on Data Engineering (ICDE)*, 2009.

[13] T. Loukopoulos, N. Tziritas, P. Lampsas and S. Lalis, "Investigating the Replica Transfer Scheduling Problem," in *18th International Conference on Parallel and Distributed Computing Systems (PDCS)*, September, 2006.

[14] T. Loukopoulos, N. Tziritas, P. Lampsas and S. Lalis, "Implementing Replica Placements: Feasibility and Cost Minimization," *in 21st International Parallel and Distributed Processing Symposium (IPDPS)*, March 2007.

[15] S. U. R. Malik , S. U. Khan, S. J. Ewen,N. Tziritas, J. Kolodziej, A. Y. Zomaya, S. A. Madani, N. Min-Allah , L. Wang, C. Xu, Q. M. Malluhi, J. E. Pecero, P. Balaji, A. Vishnu, R. Ranjan, S. Zeadally, and H. Li, "Performance Analysis of Data Intensive Cloud Systems Based On Data Management and Replication: A Survey," *Distributed and Parallel Databases*, 2015.

[16] X. Meng, V. Pappas, and L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," *IEEE Conference on Computer Communications (INFOCOM)*, 2010.

[17] J. Schroeder, A. L.P. Guedes, E. P. Duarte, "Computing the Minimum Cut and Maximum Flow of Undirected Graphs," *Technical Report*, Universidade Federal do Parana, 2004.

[18] T. Sigwele, A.S. Alam, P. Pillai, Y.F. Hu, "Energy-Efficient Cloud Radio Access Networks by Cloud Based Workload Consolidation for 5G," *Journal of Network and Computer Applications*,

vol. 78, 1-8, 2017.
[19] J. Sonnek, J. Greensky, R. Reutiman, A. Chandra, "Starling: Minimizing Communication Overhead in Virtualized Computing Platforms Using Decentralized Affinity-Aware Migration," *International Conference on Parallel Processing (ICPP)*, 2010.
[20] L. Tong, Y. Li, W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing, *" IEEE International Conference on Computer Communications (INFOCOM)*, 2016.
[21] N. Tziritas, M. Koziri, A. Bachtsevani, T. Loukopoulos, S. U. Khan, G. Stamoulis, C.-Z. Xu, "Data Replication and Virtual Machine Migrations to Mitigate Network Overhead in Edge Computing Systems," IEEE Transactions on Sustainable Computing, 2017.
[22] N. Tziritas, T. Loukopoulos, P. Lampsas and S. Lalis, "Using Multicast Transfers in the Replica Migration Problem: Formulation and Scheduling Heuristics," *in 15th International Euro-Par Conference (EUROPAR)*, August 2009.
[23] N. Tziritas, T. Loukopoulos, P. Lampsas and S. Lalis, "Formal model and scheduling heuristics for the replica migration problem," *in 14th International Euro-Par Conference (EUROPAR)*, August 2008.
[24] N. Tziritas, S. U. Khan, C.-Z. Xu, J. Hong, "An Optimal Fully Distributed Algorithm to Minimize the Resource Consumption of Cloud Applications", *in 18th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, December 2012.
[25] N. Tziritas, S. U. Khan, C.-Z. Xu, T, Loukopoulos,S. Lalis, "On Minimizing the Resource Consumption of Cloud Applications Using Process Migrations," *Elsevier Journal of Parallel and Distributed Computing*, vol. 73, no. 12, 1690-1704, 2013.
[26] N. Tziritas, S. U. Khan, T. Loukopoulos, S. Lalis, C.Z. Xu, P. Lampsas, "Single and Group Agent Migration: Algorithms, Bounds, and Optimality Issues," *IEEE Transactions on Computers*, vol. 63, no. 12, 3143-61, 2014.
[27] N. Tziritas, T. Loukopoulos, P. Lampsasand S. Lalis, "Using Multicast Transfers in the Replica Migration Problem: Formulation and Scheduling Heuristics," *in 15th International Euro-Par Conference (EUROPAR)*, 2009.
[28] N. Tziritas, C.-Z. Xu, T. Loukopoulos, S. U. Khan, Z. Yu, "Application-aware Workload Consolidation to Minimize both Energy Consumption and Network Load in Cloud Environments," *IEEE International Conference on Parallel Processing (ICPP)*, 2013.
[29] S. Verboven, K. Vanmechelen, J. Broeckhove, "Network Aware Scheduling for Virtual Machine Workloads with Interference Models," *IEEE Transactions on Services Computing*, 8(4), 617-629, 2015.
[30] J. Winick, S. Jamin, "Inet-3.0: Internet Topology Generator," Technical Report CSE-TR-456-02, *University of Michigan*, 2002.
[31] H. H. Yang, D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning," *The Best of ICCAD*, 521-534, 2003.
[32] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, J. L. Hellerstein, "Dynamic Service Placement In Geographically Distributed Clouds," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2013.
[33] http://hama.apache.org/
[34] http://hadoop.apache.org/
[35] http://home.web.cern.ch/about/experiments/cms
[36] Network Simulator2 (ns2), http://www.isi.edu/nsnam/ns/
[37] http://www.genome.gov/
[38] https://www.humanbrainproject.eu/