Data Replication and Virtual Machine Migrations to Minimize Network Overhead

(Moving data to computation or computations to data)

by Georgia Troumpoutza
and
Areti Mpachtsevani

A dissertation submitted to the
University of Thessaly

in partial fulfillment to the requirements of the degree of

Diploma of Science in
Computer and Communication Engineering

Accepted on the recommendation of:

Advisor : Georgios Stamoulis, Professor
Co-Advisor: Athanasios Loukopoulos, Lecturer

VOLOS, GREECE
JULY 2015

This page intentionally left blank

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisors, Dr. Georgios Stamoulis and Dr. Athanasios Loukopoulos for the great collaboration and their guidance in all the time of writing this thesis.

ABSTRACT

Although several virtual machine (VM) placement algorithms have been proposed and studied in the literature, little research has been done so far on capturing and minimizing the network overhead when combining the VM assignment problem on clusters with the problem of replicating the files accessed by the VMs hosted within the system. We also study the aforementioned problem when clusters have limited storage and computing capacity. We propose an algorithm based on hyper-graph partitioning to solve the aforementioned problem when there are no computing and storage capacity constraints on clusters. The proposed algorithm is extended to capture the storage and computing capacity constraints on clusters within the system. An experimental evaluation is given to compare the behavior of the proposed algorithm. The results shown that the proposed algorithm yields a network overhead reduction of up to 50% compared to state-of-the-art algorithms found in the literature.

ΠΕΡΙΛΗΨΗ

Παρ'όλο που αρκετοί αλγόριθμοι τοποθέτησης ιδεατών μηχανών (VM) έχουν προταθεί και μελετηθεί στη βιβλιογραφεία, ελάχιστη έρευνα έχει γίνει μέχρι στιγμής πάνω στην αποτύπωση και ελαχιστοποίηση της επιβάρυνσης δικτύου όταν συνδυάζουμε το πρόβλημα ανάθεσης VM σε συστάδες με το πρόβλημα της αντιγραφής αρχείων που προσπελαύνονται απο τις VM που φιλοξενούνται στο σύστημα. Επιπλέον μελετούμε το προαναφερθέν πρόβλημα όταν οι συστάδες έχουν μειωμένη χωρητικότητα και υπολογιστική ικανότητα.Προτείνουμε έναν αλγόριθμο βασισμένο στη διαίρεση υπεργράφων για να επιλύσουμε το προαναφερθέν πρόβλημα όταν δεν υπάρχουν περιορισμοί υπολογιστικής και χωρητικής ικανότητας στις συστάδες.Ο προτεινόμενος αλγόριθμος επεκτείνεται για να καλύψει τους περιορισμούς υπολογιστικής και χωρητικής ικανότητας των συστάδων στο σύστημα.Δίνεται μια πειραματική εκτίμηση για τη συγκριση της συμπεριφοράς του προτεινόμενου αλγόριθμου.Τα αποτελέσματα δείχνουν οτι ο προτεινόμενος αλγόριθμος αποφέρει μείωση επιβάρυνσης δικτύου μέχρι και 50% συγκριτικά με αλγόριθμους τελευταίας λέξης της τεχνολογίας που βρίσκονται στη βιβλιογραφεία.

TABLE OF CONTENTS

1. Introduction and background

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 [34] at CERN (European Organization for Nuclear Research), the Human Genome Project [37] the Sloan Digital Sky Survey experiment [40] and the Human Brain Project [39]. 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 datasets.

Many data- and compute-intensive middleware solutions do exist in the literature, namely, Hadoop [33], Apache HAMA [32], Stork [10], Pegasus [38], Swift [36] and StorkCloud [9]. A general approach of those initiatives is to move computations close to data. The above is corroborated by the Hadoop community [33] 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.*"

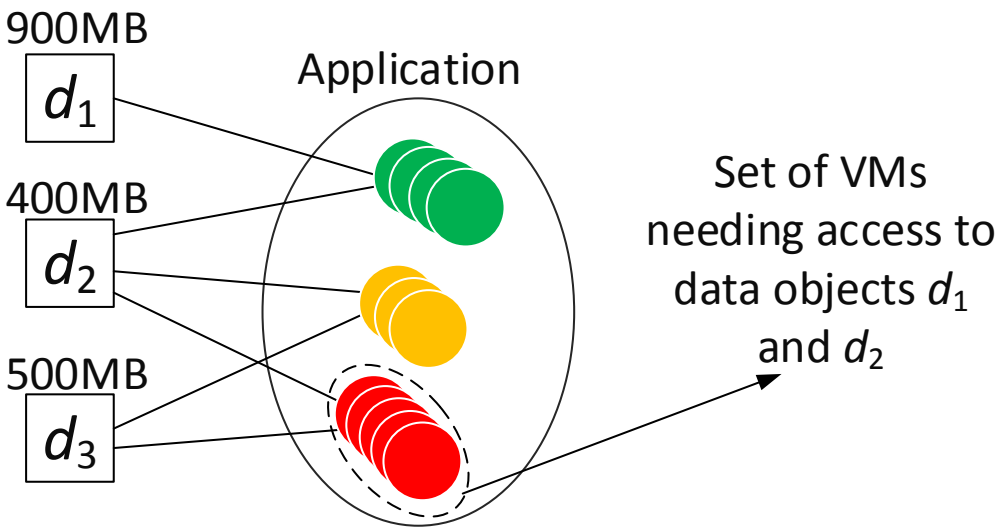

Figure 1. Application of VMs and their access to data objects

In the aforementioned example, there were no communication dependencies between VMs. However, it is of paramount importance to also take into account communication dependencies between VMs, because otherwise we may result in network and application performance degradation. It is evident that when considering communication between VMs, decisions for VM placement and data replication become more complicated. The problem belongs to the NP-Complete class for general structured networks, even when there are no decisions for replicating data [3].

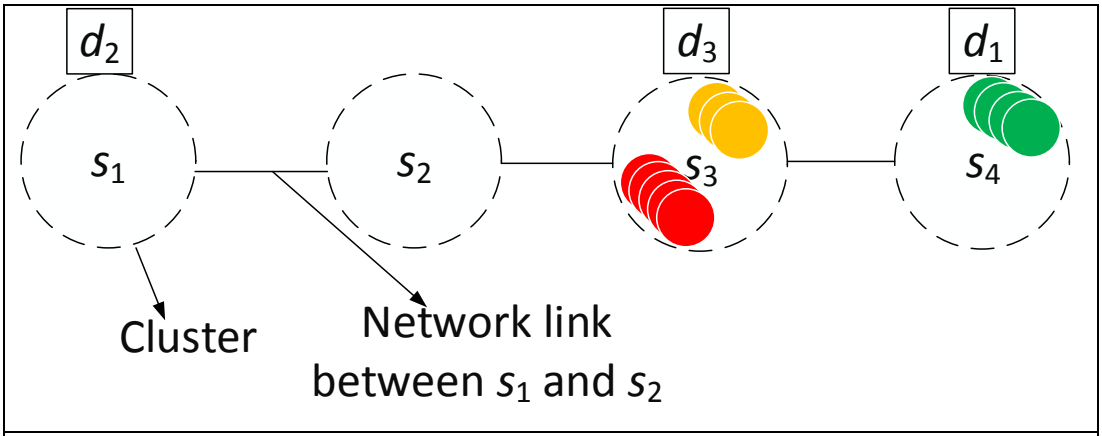

Figure 2. Application placement without replication

In this thesis, we 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. The reason that we focus in tree-structured networks is justified by the following: **(a)** It is well known that Fat-tree topology is a network topology that has been widely adopted in clouds [2]. **(b)** Elastic tree is also a network-wide optimizer that chooses a set of network elements that must be active such that the network topology forms a tree [5]. It has been shown that elastic-tree can save up to 50% of network energy. Therefore, it is evident the usefulness of algorithms working in tree-structured networks for cloud environments.

The rest of this thesis is organized as follows. In Section 2, we describe our system model, and give respective formulations of the problem we tackle. In Section 3, we give an algorithm based on hyper-graph partitioning for solving the problem without considering storage and computing capacity constraints on clusters. In Section 4, we extend the proposed algorithm to solve the problem when considering computing and storage capacity constraints on clusters. In Section 5 we evaluate the performance of the proposed algorithm against state-of-the-art algorithms found in the literature. In Section 6 we discuss related work. Finally, Section 7 concludes the thesis, identifies open issues and points towards possible future research directions.

2. System Model and Problem Formulation

This section is split into two parts. The first part includes the notations for the description of application and network structures, while the second one extends the notations for the formulation of the problem.

2.1 System model

The application is structured as a general graph with a number of V and D datasets. Let v_i and d_j denote the i^{th} VM and j^{th} dataset, 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. It must be noted that each time a VM is needed to be migrated from one cluster to another one, we transfer only its state to avoid redundant network overhead. Therefore, it is assumed that each cluster is equipped with all of the VM types. The computing requirements of i^{th} VM is captured by $r(v_i)$. The data exchanged for the access of VMs to data objects 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 data transferred from d_j towards v_i (or from d_i towards v_j, respectively); and (iii) when both i and j are more than V, then c_{ij} is always zero because there is no communication between datasets. 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.

2.2 Problem Formulation

Before proceeding to the problem formulation, we will extend the notations. The placement of VMs and data on clusters is captured by an [(V+D)×S] matrix denoted by *F*. Let f_{ix} whether the x^{th} cluster 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 data. Given a placement *F*, Eq.1 captures the network overhead incurred within the system due to the communication between VMs.

Let *R*(*j*) encode the clusters hosting replicas of j^{th} data object. The variable φ_{xj}(captured by Eq. 2) denotes the minimum distance among the distances between a cluster s_x and clusters hosting the j^{th} data object. Eq. 3 splits into two cases: (a) if a cluster s_x hosts one or more VMs that need access to j^{th} data object, then the network overhead equals the bytes of j^{th} data object multiplied by the distance between s_x and the nearest cluster hosting the j^{th} data object; (b) if a cluster s_x does not host any VM needing access to j^{th} data object, then there is no network overhead. Eq. 4 signifies the total network overhead due to the needs of VMs to access data.

$\Delta 1(F) = \sum_{i=1}^{V}\sum_{j=1}^{V}\sum_{x=1}^{S}\sum_{y=1}^{S} c_{ij} \times f_{ix} \times f_{jy} \times w_{xy}$	(1)
$\varphi_{kx}(F) = \min_{\forall y \in R(j)} w_{xy}$	(2)
$\lambda_{kx}(F) = \begin{cases} \frac{\sum_{i=1}^{V} f_{ix} \times c_{i\kappa} \times \varphi_{kx}(F)}{\sum_{i=1}^{V} f_{i\chi}}, min\left(1, \sum_{i=1}^{V} f_{ix} c_{ik}\right) = 1 \\ 0 \qquad\qquad\qquad\qquad , min\left(1, \sum_{i=1}^{V} f_{ix}\, c_{ik}\right) = 0 \end{cases}$	(3)
$\Delta 2(F) = \sum_{k=V+1}^{V+D}\sum_{x=1}^{S} \lambda_{kx}(F)$	(4)
$\Delta(F) = \Delta 1(F) + \Delta 2(F)$	(5)

$\sum_{i=1}^{V} f_{ix} \times \delta(v_i) + \sum_{j=1}^{D} f_{D+j,x} \times \delta(d_j) \leq \delta(s_x), 1 \leq x \leq S$	(6)
$\sum_{i=1}^{V} f_{ix} \times r(v_i) \leq r(s_x), 1 \leq x \leq S$	(7)

The total network overhead within the network is captured by Eq. 5, which equals the sum of Eq. 1, Eq. 4. The constraint expressed by Eq. 6 signifies that the total storage space of a cluster cannot be exceeded by the sum of the storage requirements of data and VMs hosted by the respected cluster. On the other hand, Eq. 7 denotes that the total computing capacity of a cluster cannot be exceeded by the sum of the computing requirements of VMs hosted by the respective cluster.

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 clusters, and a set of clusters along with their computing capacities, find an initial assignment of VMs onto clusters as well as a data object replica scheme (recorded by F) such that the total network overhead expressed by Eq. 5 is minimized while the computing capacity constraints (Eq. 9 and Eq. 10, respectively) are not violated.*

3. Hypergraph Partitioning Algorithm without Considering Storage and Computing Capacity Constraints

Firstly, we will reduce the problem to a hypergraph partitioning problem. In the sequel, to address the problem we will employ a hypergraph partitioning technique that is based on the maximum flow problem.

3.1 Reducing the Problem to a Hypergraph Partitioning Problem of two clusters

The application graph along with its access on data is transformed in a hyper-graph as follows. For each data object, we find the set of VMs needing access on it. The aforementioned set of VMs and the respective data object form a hyper-edge. The weight of the hyper-edge equals the size of the corresponding data object. Such a weight signifies the 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. The above is justified by that fact that when splitting a hyper-edge into two parts, then the corresponding data object must be replicated towards the side of VMs that are not co-located with it. Each regular edge between two VMs denotes the data exchanged between the VMs. When we split 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. Such a partition encodes the VM assignment and data replication onto two clusters. By finding the minimum cut, we result in an assignment of VMs as well as a replication of data onto two clusters with the minimum network overhead.

The aforementioned is illustrated through the following example. Consider the application shown in Fig. 3. The application consists of four VMs that 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 access d_j. An edge between two VMs represents the exchange of data between the respective VMs, with the weight of the edge signifying the size of data

being exchanged. Consider also the network shown in Fig. 4 consisting of two clusters (s_1, s_2), with s_1 hosting d_1, while s_2 hosting d_2.

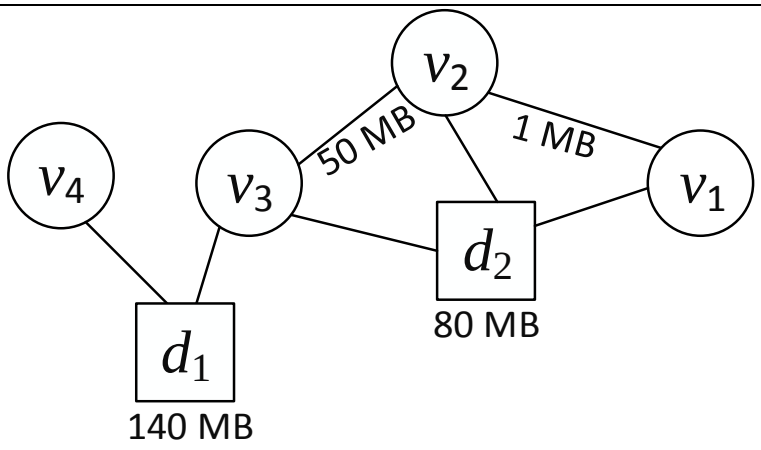

Figure 3. Application and data graph

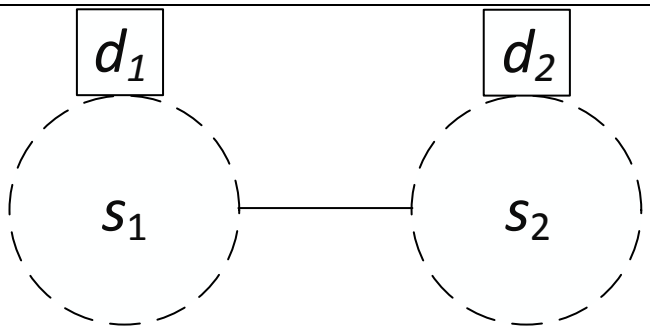

Figure 4. Network graph

In the sequel, we find all of the hyper-edges within the application graph. The hyper-edges are shown in Fig. 5. For each data object, we create a hyper-edge that contains 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 least 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 blue 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_4 and v_5 access d_1 at different points in time, then v_4 and v_5 must separately access data

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 blue 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.

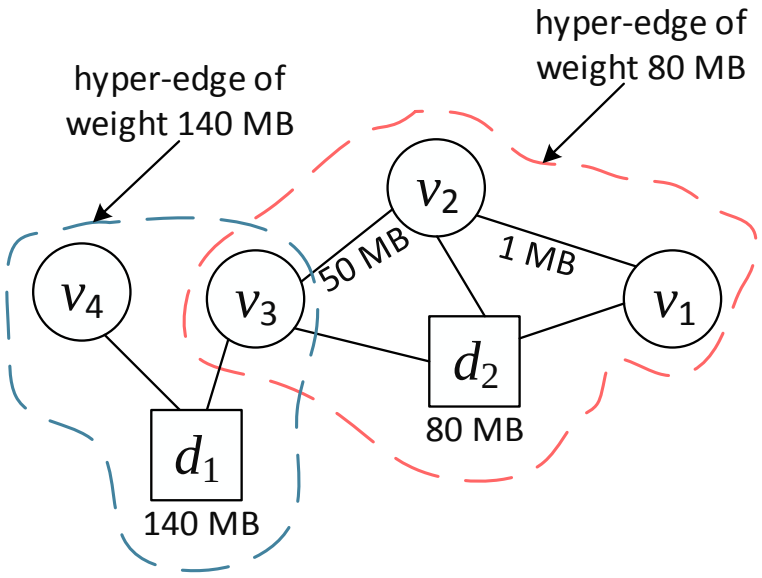

Figure 5. Hyper-edges

Therefore, to minimize the network overhead, we must bi-partitioning the hyper-graph shown in Fig. 5. 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 red hyper-edge is split, 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. 6. It must be stressed that by not applying the concept of hyper-edges, then the partitioning of the application would be as follows. The VMs v_4 and v_3 would have been assigned onto s_1, while the rest ones onto s_2, incurring a network overhead of 130 (80+50).

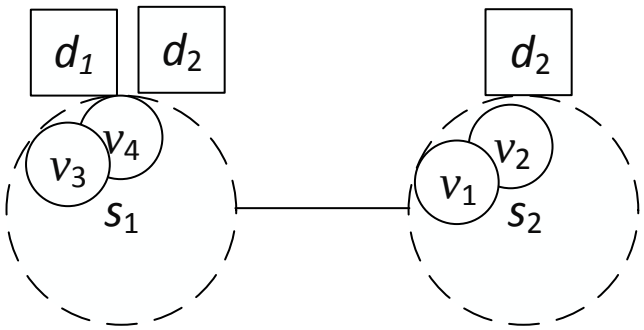

Figure 6. Final VM placement and data replicas

3.2 Initial Assignment of VMs and Data Replicas

In this section we consider that the VMs have not been assigned to any cluster within the system. Therefore we examine the initial VM placement as well as the data object replicas. According to [2] the problem of hyper-graph bi-partitioning can be solved optimally by reducing to the problem of finding a cut of minimum capacity. The reduction of the problem takes place through the transformation of hyper-edges into regular edges.

3.2.1. Hyper-edge transformation into a set of regular edges

Consider a hyper-edge *e* of weight *w* 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: (a) we add two auxiliary vertices (u_x and u_y) as well as a bridging edge of weight *w* between them; (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.

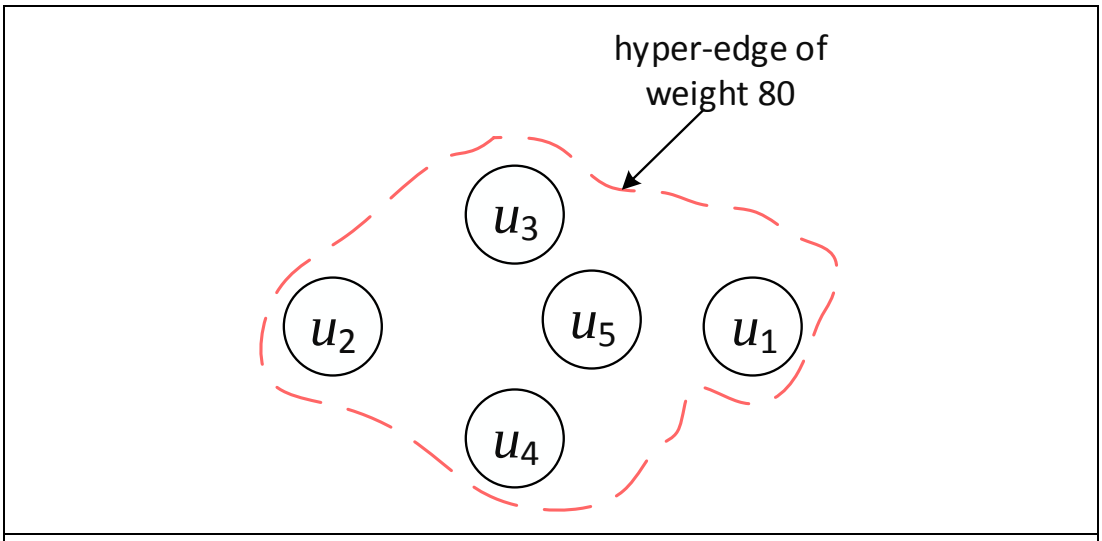

Figure 7. Hyper-edge of weight 80 with five vertices

An example is shown below where a hyper-edge consisting of five vertices and its weight equaling 80 (Fig. 7) is transformed into a set of regular edges (Fig. 8). 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.

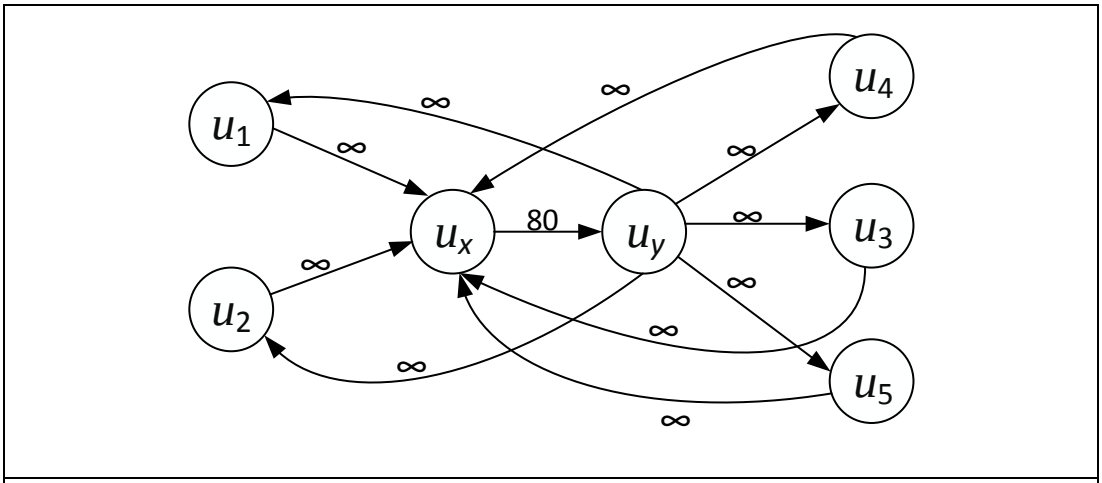

Figure 8. Hyper-edge transformation

3.2.2 Solving the problem with two clusters

Given an application that needs access to some data objects hosted by two clusters, then the VM placement and data replication problem is solved by employing the {s-t} minimum cut algorithm as follows: (i) add two vertices s_1 and s_2, representing the corresponding two clusters; (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, we use the name of the data object (d_i) belonging to the respective edge; (iii) add a directed edge of $\delta(d_i)$ weight originating from d_i's hosting cluster and ending on d_i, as well as an edge of infinite weight originating from u_y and ending on d_i's hosting cluster; (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 [2] , 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 to s_1, while the rest ones are assigned 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. It must be noted that the minimum cut is not affected by incorporating in the minimum cut graph regular edges residing within hyper-edges.

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 the data by the VMs are shown in Fig. 5. As can be seen, there are two hyper-edges of weight 140 and 80. Assume that there are two clusters (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 blue hyper-edge is transformed into the set of vertices $\{d_1, v_y, v_4, v_5\}$ with regular edges, while the red one is transformed into the set of vertices $\{d_2, v_{y'}, v_1, v_3, v_4\}$ 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_3, as well as an edge between v_3 and v_4. The resultant minimum cut graph is shown in Fig. 9.

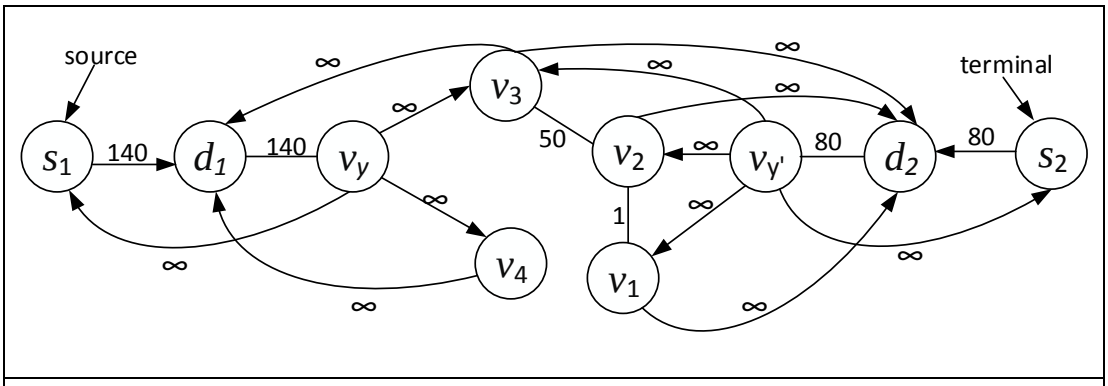

Figure 9. Minimum cut graph

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 $v_{y'}$ and d_2, with the cost of cut being 80 MB. Specifically, we have chosen the augmenting path $\{s_1, d_1, v_y, v_4, d_2, v_{y'}, s_2\}$ 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. 10. From the final residual graph, we observe that the set of vertices that are reachable from s_1 is $\{d_1, d_2, v_y, v_1, v_3, v_4, v_5\}$. Consequently, the only vertex that is reachable from s_1 is $v_{y'}$. Therefore, all of the VMs are assigned on s_1, with d_2 being replicated on s_1. Note that s_2 continues hosting d_2. By performing the aforementioned assignment of VMs and data replicas, we result in a total network overhead of 80 MB.

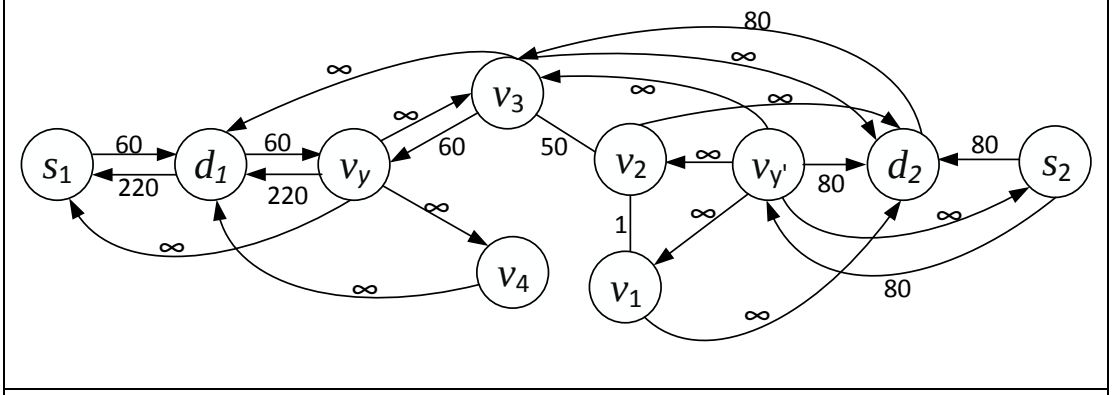

Figure 10. Final residual graph

3.2.3 Extending the solution to a tree of clusters

In this section we extend the aforementioned algorithm for the problem of two clusters to a more general algorithm for a tree of clusters. The pseudocode of the extended algorithm (called HPA) is shown in Table I. Below we give an explanation of the pseudocode: **(a)** All of the clusters are marked as unexplored (line 1). In the sequel, a cluster is randomly chosen and marked as explored (line 2). All of the VMs are temporarily assigned on the aforementioned cluster (line 3). **(b)** Choose an unexplored cluster s_i that is 1-hop away from any explored cluster 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 cluster is closer to s_i, or otherwise s_j (line 8-9). 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, with the vertex u_x being declared as d_k; add a directed edge of $\delta(d_k)$ weight originating from either s_i provided that s_i is closer to the d_k's hosting cluster against s_j, or otherwise s_j, and ending on d_k; add also an edge of infinite weight originating from u_y and ending on s_i provided that s_i is closer to the d_i's hosting cluster against s_j, otherwise on s_j (line 12-15). **(e)** For any edge e between a VM v_g hosted by s_j and a VM v_f not hosted by it, we draw an edge between v_g and $s_{j'}$ with the same weight as that of e. The above is because if v_g is assigned on s_i, then it distances itself away from v_f (line 18-20). **(f)** Apply the maximum flow minimum cut algorithm to temporarily re-assign the VMs onto clusters as explained in Section 3.2.2 (line 21). **(g)** If there is any v_i accessing a d_k, with the v_i distancing itself away from the cluster hosting the original copy of d_k, replicate d_k at the cluster hosting v_i (line 22) The steps from (b) to (g) are repeated until there is no unexplored cluster. **(h)** The last step is to revoke a replication in case a decision has been made for replicating a data object d_k at a cluster s_i, provided that there is no VM hosted by s_i that accesses d_k (line 24).

Table I. Pseudocode of HPA	
1:	mark all of the clusters as unexplored
2:	choose randomly a cluster and mark it as explored
3:	assign all of the VMs on the aforementioned cluster
4:	**for each** unexplored cluster s_i that is 1-hop away from any explored cluster s_j
5:	mark s_i as explored
6:	draw a new min-cut graph and add s_i and s_j
7:	**for each** data object d_k belonging to D

8:	identify a hyper-edge/regular edge e for d_k considering only VMs assigned on s_j
9:	**if** e is a regular edge
10:	draw an edge between the respective VM and s_i/s_j
11:	**else**
12:	transform the hyper-edge into a set of regular edges as explained in §3.2.1
13:	declare u_x as d_k
14:	add a directed edge of infinite weight from s_i/s_j towards d_k
15:	add a directed edge of infinite weight from u_y towards s_i/s_j
16:	**end if**
17:	**end for**
18:	**for each** edge e between a VM v_g hosted by s_j and a VM v_f not hosted by it
19:	draw an edge between v_g and s_j with the same weight as that of e
20:	**end for**
21:	apply max-flow min-cut algorithm to re-assign VMs according to §3.2.2
22:	if there is any v_i accessing a d_k, with the v_i distancing itself away from the cluster hosting the original copy of d_k, replicate d_k at the cluster hosting v_i.
23:	**end for**
24:	if there has been a decision for replicating d_k at s_i, but there is no VM hosted by s_i that accesses d_k, then revoke the respective replication

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). As can be seen from Fig. 12, the network is consisted of three clusters s_1, s_2, and s_3 hosting d_1, d_2, and d_3, respectively.

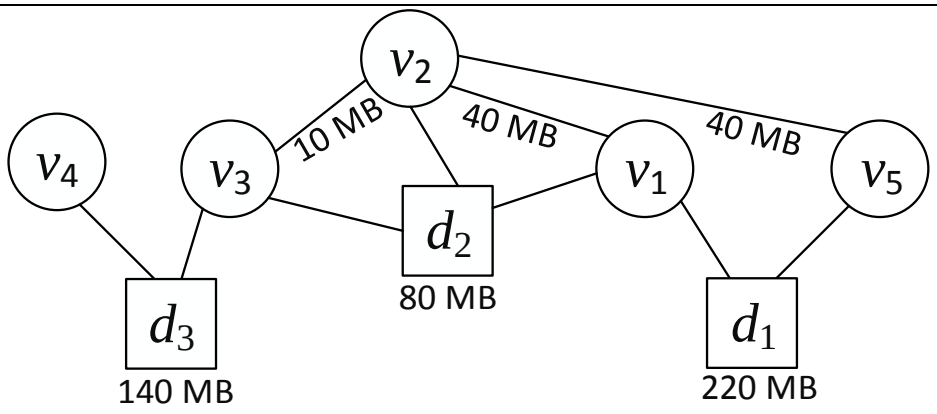

Figure 11. Application structure

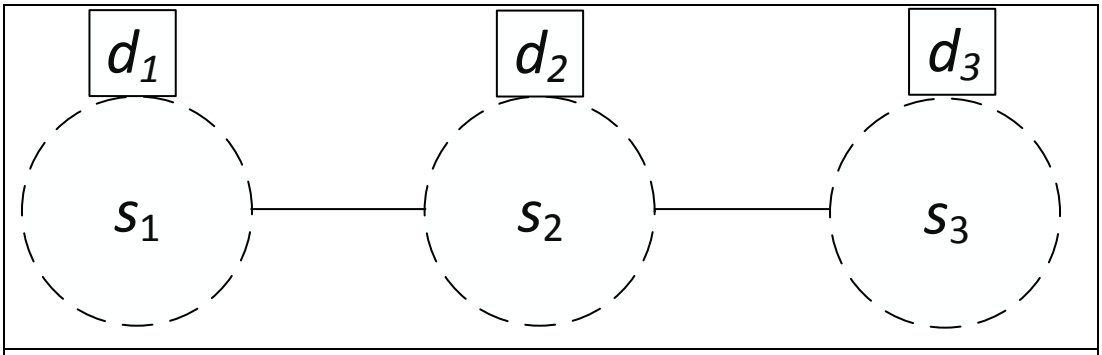

Figure 12. Network structure

The algorithm begins by marking all of the clusters 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 objects d_1, d_2, and d_3 (see Fig. 13).

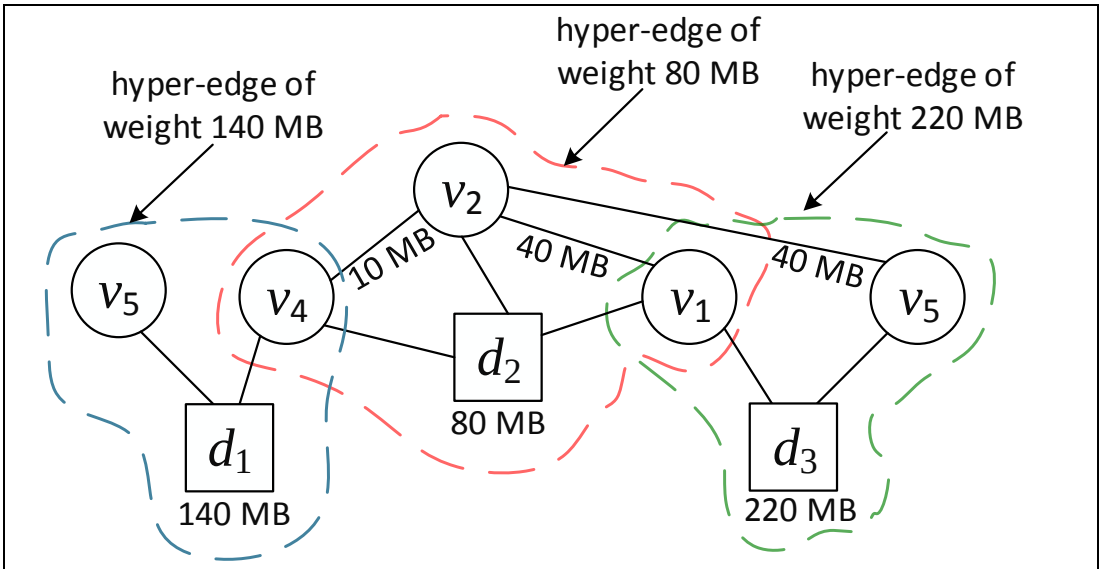

Figure 13. Hyper-edges identified when considering s_1 and s_2

The transformation of each hyper-edge into a set of regular edges takes place according to Section 3.2.1. Because the hosting cluster of d_1 and d_2 is s_1 and s_2, respectively, an infinite edge is drawn from s_1 towards d_1 and another one from s_2 towards d_2. On the other extreme, the hosting cluster of d_3 is closer to s_2 than s_1, as a result an edge of infinite weight is draw from s_2 towards d_3.

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. 14.

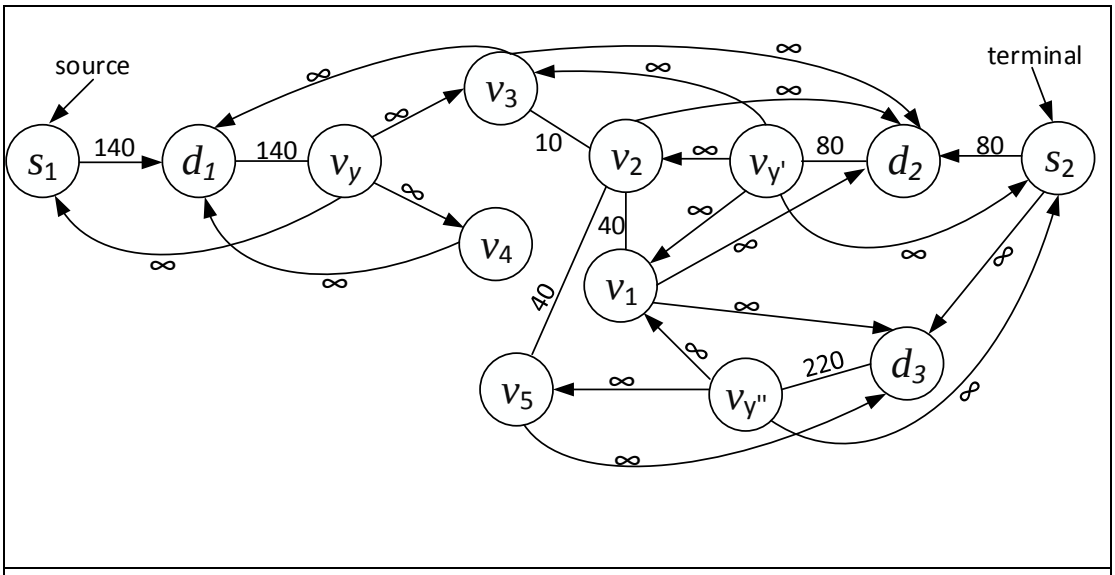

Figure 14. Min-cut graph when considering s_1 and s_2

After executing the maximum flow minimum cut algorithm regarding the graph of Fig. 13, we result in the residual graph shown in Fig. 15. It is observed that the minimum cut equals 90 (80+10), with the reachable set of vertices from s_1 being {d_1,v_y, v_3, v_4, d_2}. Therefore, v_3 and v_4 are assigned onto s_1, while d_2 is replicated at s_1. On the other extreme, v_1, v_2, and v_5 are assigned onto s_2, while there is no decision for replicating d_3 at s_2. According to the line 22 of HPA, there is no decision for replicating d_3 at s_2, because there is no VM accessing d_3 and distancing itself away from s_3 (the host of the original copy of d_3). The aforementioned assignment and replication decisions are shown in Fig. 16. It must be noted that they are temporary because s_3 has not been explored yet.

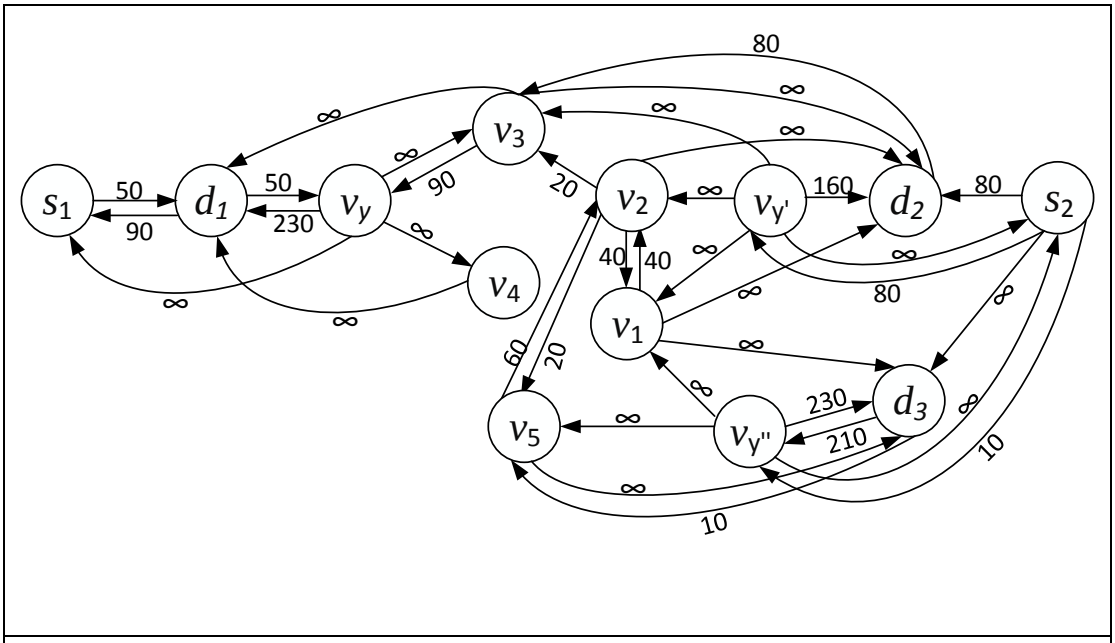

Figure 15. Residual graph when considering s_1 and s_2

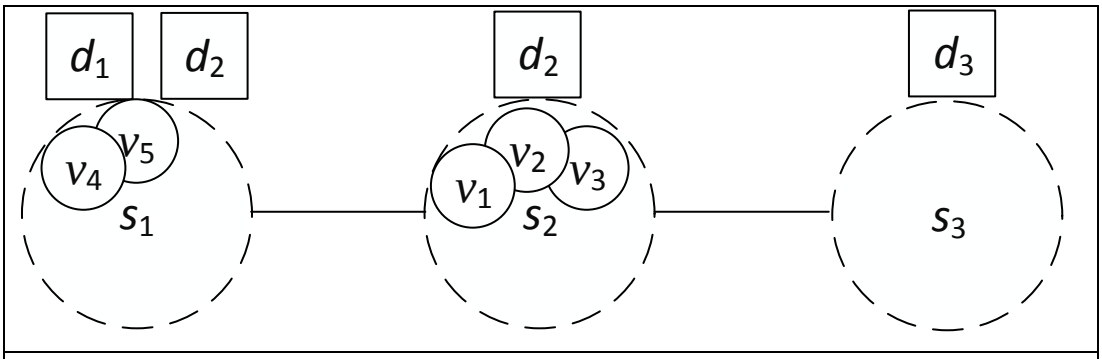

Figure 16. Temporary VM assignments and data replications

In the sequel, s_3 is the next (and last) cluster 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. 17. The transformation of each hyper-edge into a set of regular edges takes place according to Section 3.2.1. Because the hosting cluster of d_2 is s_2, an infinite edge is drawn from s_2 towards d_2. On the other extreme, the hosting cluster of d_3 is s_3, as a result an edge of infinite weight is draw from s_3 towards d_3. Due to the fact that there exists an edge between v_2 (hosted by s_2) and v_4 (hosted by s_1), an edge is drawn between v_2 and s_2. The reason we chose s_2 and not s_3 as the one end of the edge is that the hosting cluster 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. 18.

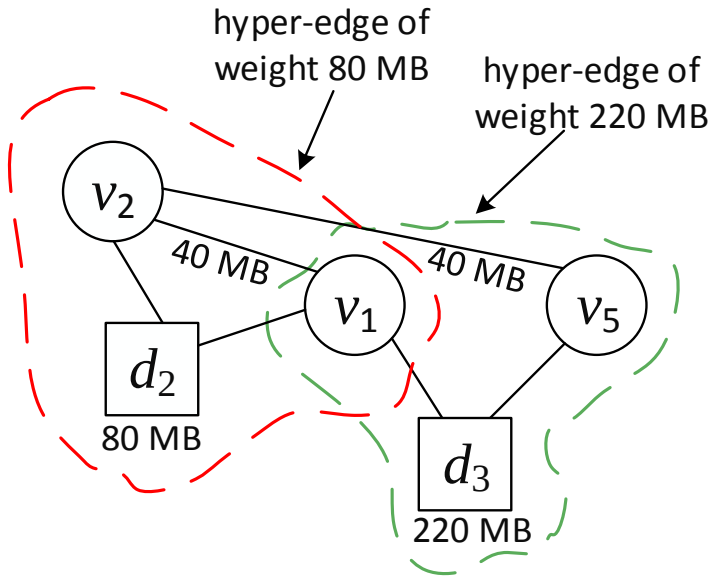

Figure 17. Hyper-edges identified when considering s_2 and s_3

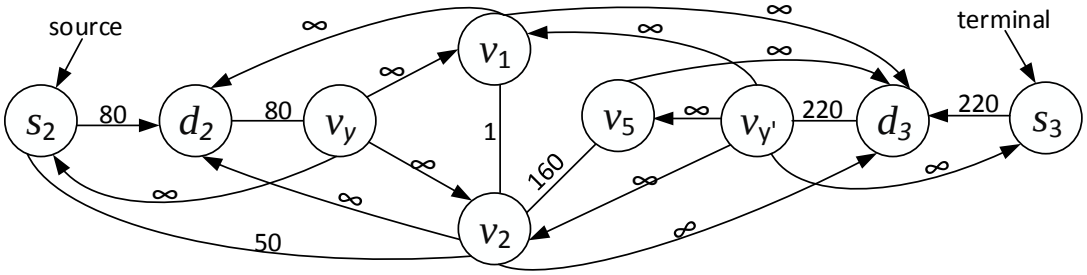

Figure 18. Min-cut graph when considering s_2 and s_3

By applying the minimum cut maximum flow algorithm for the graph shown in Fig. 18, we result in the residual graph depicted in Fig. 19. As can be seen, the minimum cut equals 90 (80+10), with the reachable set of vertices from s_2 being $\{\emptyset\}$. As a result, the set of vertices that are reachable from s_3 is $\{d_2, v_y, v_1, v_2, v_5, v_{y'}, d_3\}$. Consequently, the set of VMs $\{v_1, v_2, v_5\}$ is assigned onto s_3, with d_2 being replicated at s_3. The final VM assignments and data replications are shown in Fig. 20, with the total network overhead being 180 MB. On the other extreme, by not exploiting the hyper-edge partitioning technique with data replication and considering only regular edges, the minimum cut algorithm would result in the following assignment.

The VMs v_3 and v_4 would have been assigned onto s_1, v_2 onto s_2, while v_1 and v_5 onto s_3. Such an assignment would incur a total network overhead of 250 MB.

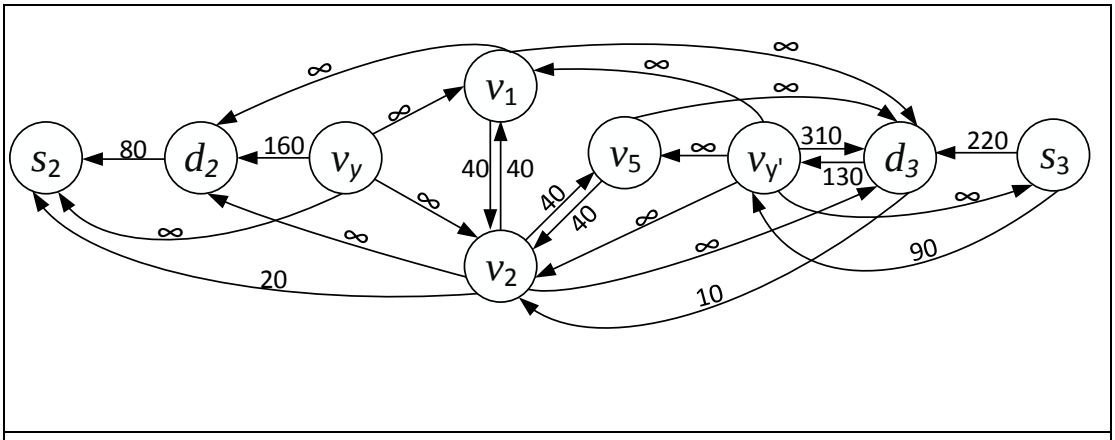

Figure 19. Residual graph when considering s_2 and s_3

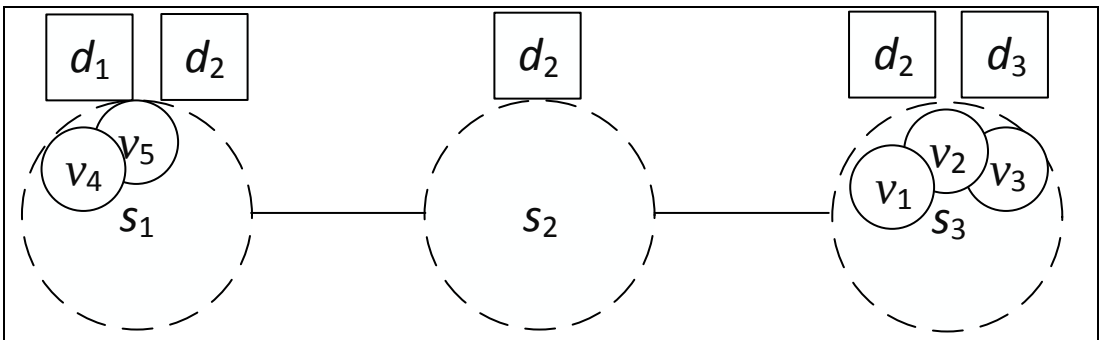

Figure 20. Final VM assignment and data replication

4. 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.

4.1 Extending HPA for the Two Cluster Case when Considering Capacity Constraints

In this section, the VM assignment and data object replication problem is solved for a system consisting of two clusters (s_1, s_2). We assume that there is at least one assignment of VMs onto clusters such that the computing capacity of both clusters is not violated. 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 capacity violation in any cluster within the system, then the replication takes place as dictated by the solution for the un-capacitated case. Otherwise, we replicate in an iterative fashion the object that its replication reduces the network overhead as much as possible.

Step 2. The problem is solved again by replacing a hyper-edge with normal edges iff its involved data object is not able to be replicated according to step 1. Specifically, the replacement takes place as follows. For each VM participating in such a hyper-edge we add a normal edge of weight equal to the involved data object between the respective VM and the cluster hosting the corresponding data object. After obtaining the solution of this step, we investigate whether any cluster'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 cluster (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 clusters 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 clusters. 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 least network overhead is decided to be performed. The aforementioned re-assignment process iterates itself until either there is no violation of s_1's computing capacity or there is no VM re-assignments that does not violate s_2's computing capacity. In case of violation of s_2's computing capacity, we run knapsack twice as follows. In terms of the first knapsack instance, knapsack plays the role of s_1, with its computing capacity reflecting knapsack's size. The set of VMs hosted by both s_1 and s_2 plays the role of knapsack objects, with their computing requirements representing the weight of knapsack objects. The benefit of knapsack objects is assumed of one unit. Regarding the second knapsack instance, knapsack plays the role of s_2, while the knapsack objects that were not assigned on first knapsack instance represent the knapsack objects of the second knapsack instance.

VM re-assignment impact. The impact of re-assigning a VM v_i from s_x to s_y is the difference, in terms of network overhead, between hosting the corresponding VM on s_x and s_y. The aforementioned impact is expressed by Eq. 11 and split into two components:

1) The first component (expressed by Eq. 8) concerns the difference (when assigning v_i onto s_y and s_x) in network overhead due to the data exchanged between v_i and the VMs it communicates with. Note that F^{old} and F^{new} captures the placement before and after the re-assignment of v_i from s_x to s_y. Therefore, it holds that $f_{ix}^{old} = 1$ and $f_{iy}^{new} = 1$.
2) The second component (expressed by Eq. 9) represents the difference (when assigning v_i onto s_y and s_x) in network overhead due to the need of v_i to access the data involved in hyper-edges. The variable H_{ik} equals one if v_i is contained

in the hyper-edge that involves d_k, otherwise equals zero. Note that when calculating the impact in network overhead when re-assigning v_i from s_x to s_y the following take place: **(a)** if H_{ik} equals 1, v_i was the only VM hosted by s_x that needed access to d_k which was also hosted by s_x, then d_k is deleted from s_x (i.e., $f_{kx}^{new} = 0$); and **(b)** if H_{ik} equals 1 and s_y has available storage capacity to host d_k, then s_y will host d_k (i.e., $f_{ky}^{new} = 1$).

$I_{ixy}^{VM} = \sum_{j=1}^{V} c_{ij} \times (1 - f_{jy}^{new}) - \sum_{j=1}^{V} c_{ij} \times (1 - f_{jy}^{old})$	Eq. 8
$I_{ixy}^{H} = \sum_{k=V+1}^{V+D} \lambda_{ky}(F^{new}) \times H_{ik} - \sum_{k=V+1}^{V+D} \lambda_{kx}(F^{old}) \times H_{ik} + \sum_{k=V+1}^{V+D} \delta(d_k) \times H_{ik} \times (1 - f_{ky}^{old})$	Eq. 9
$I_{ixy}^{Total} = I_{ixy}^{VM} + I_{ixy}^{H}$	Eq. 10

4.2. Extending HPA for the Tree Cluster Case when Considering Capacity Constraints

In this section, the VM assignment and data object replication problem is solved for a system consisting of *N* clusters structured as a tree. We assume that there is at least one assignment of VMs onto clusters such that there is no violation in their computing capacities. The procedure is identical with that of Section 4.1 up to step 2. Regarding step 3 the following take place. For each cluster that its computing capacity is violated, we attempt to find a re-assignment of its VMs towards other clusters 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. In case of the former case, the algorithm terminates. Otherwise, the knapsack is solved (for more information, the reader is referred to Section 4.1 in step 3) between all possible cluster pairs until there is no computing capacity violation in any cluster.

VM re-assignment impact. The impact of re-assigning a VM v_i from s_x to s_y is the the same as that described in Section 4.1, with the differences being that here **(a)** there are also other clusters 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. The aforementioned impact is expressed by Eq. 13 and split into two components:

1) The first component is the same as its counterpart depicted in Eq. 8, with the difference being that here the distance between the clusters hosting communicating VMs varies according to *W*.
2) The second component is a little bit different against its counterpart depicted in Eq. 9. Here we take into account all of the servers within the system instead of only s_x and s_y. The above is because when deleting or creating a replica, there is an impact in network overhead regarding the VMs contained in the hyper-edge that involves the respective data object.

$I'^{VM}_{ixy} = \sum_{j=1}^{V}\sum_{z=1}^{S} c_{ij} \times f_{jz} \times w_{yz} - \sum_{j=1}^{V}\sum_{z=1}^{S} c_{ij} \times f_{jz} \times w_{xz}$	Eq. 11
$I'^{H}_{ixy} = \sum_{k=V+1}^{V+D}\sum_{z=1}^{S} \lambda_{kz}(F^{new}) \times H^{y}_{ik} - \sum_{k=V+1}^{V+D}\sum_{z=1}^{S} \lambda_{kz}\left(F^{old}\right) \times H^{x}_{ik} + \sum_{k=V+1}^{V+D} \delta(d_k) \times H_{ik} \times \left(1 - f^{old}_{ky}\right) \times \varphi_{\kappa\chi}\left(F^{old}\right)$	Eq. 12
$I'^{Total}_{ixy} = I'^{VM}_{ixy} + I'^{H}_{ixy}$	Eq. 13

5. Evaluation

The experimental evaluation has been conducted on NS-2 [35]. Five different network topologies were generated with the number of clusters being fixed at 30. The networks are generating by placing randomly the nodes in a plane of 60×60 distance units. Clusters are assumed to be in range of each other if their Euclidean distance was less than ρ distance units (with ρ being uniformly distributed between 30 and 40). The corresponding tree-based routing topology is obtained by constructing a spanning tree, whereby each pair of clusters is connected via a single path. Ten different general-structured application graphs were constructed, with the number of VMs ranging between 400 and 600 (uniformly distributed). The generation of application graphs takes place in the same way as that of network generation. The only difference is that we do not apply the part of applying the spanning tree.

The evaluation was split into two parts. The first part involved clusters without computing and storage capacity constraints, while the second one took into consideration computing and storage capacity constraints on clusters. The comparison was conducted among HPA, DRA, and DBA.

5.1 Uncapacitated Clusters

The first set of experiments was conducted without considering computing and storage capacity constraints on clusters. 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 clusters 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. 21, HPA achieves superior performance against DRA and DBA. Specifically, HPA achieves a network overhead reduction of 50% and 30% against DBA 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. On the other hand, DRA takes into consideration the migration of VMs in a grouped manner resulting in better results against DBA. The superiority of HPA is attributed to the following facts: (a) HPA considers replicating data objects, with the VMs having the option to access those objects from any cluster holding their replicas. In that way, the network overhead can be significantly reduced since a cluster can access an object from the nearest cluster holding the replica of that object instead of accessing it from the cluster holding the initial replica of the respective object. And (b) the assignment of VMs takes place in a way such that the co-located VMs needing access to the same data object can share the transfer of the corresponding data.

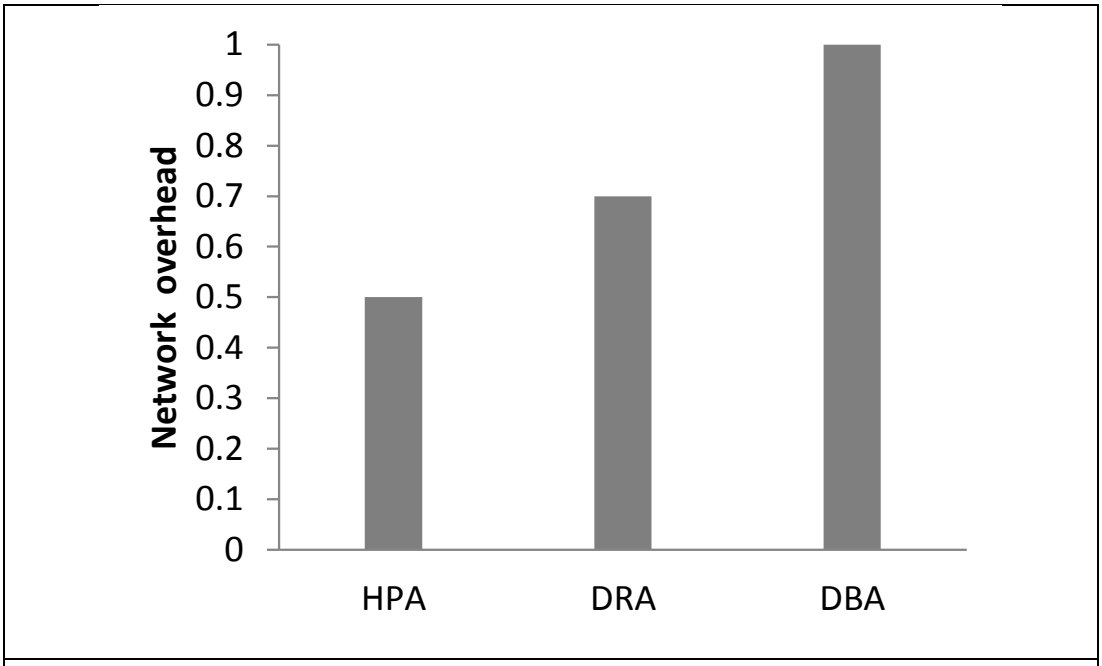

Figure 21. Network overhead

5.2. Capacitated Clusters

In this section, we conducted experiments for the comparison of HPA with DRA and DBA under computing and storage capacity constraints. Initially, the computing capacity of each cluster 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 cluster within the system from 10 to 50 VMs. As we can see from Fig. 22, HPA achieves the best performance even when the surplus computing capacity of each cluster is tight. Specifically, when the surplus computing capacity of each cluster equaled 10 VMs, HPA yielded a network overhead reduction of roughly 7% and 4% against DBA and DRA, respectively. On the other extreme, when the surplus computing capacity of each cluster was relaxed, the HPA achieved a bigger network overhead reduction against DBA and DRA.

Specifically, when the surplus computing capacity of each cluster was equal to 50, the network overhead reduction of HPA compared to DBA and DRA was 42% and 20%, respectively.

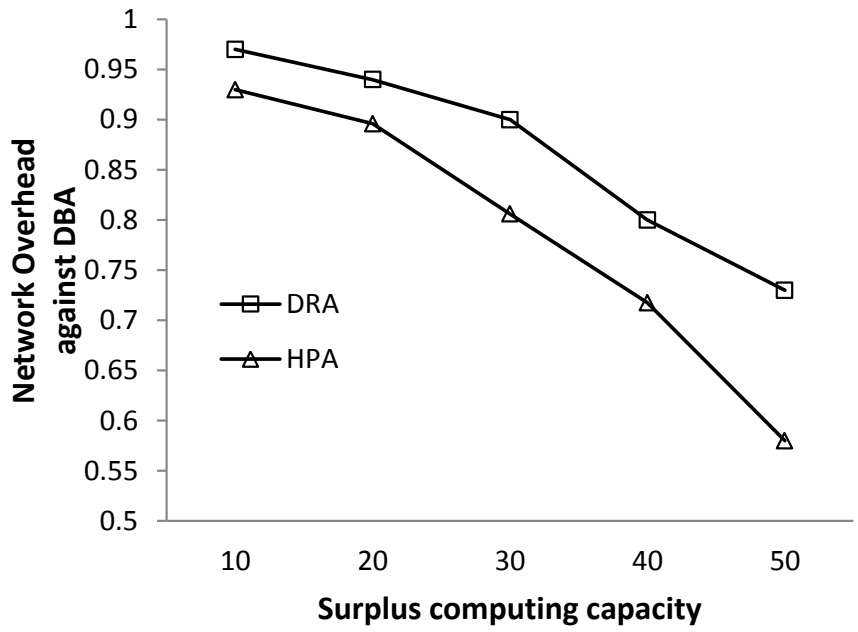

Figure 22. Network overhead when varying surplus computing capacity of each cluster within the system.

The second set of experiments was conducted to investigate the behavior of HPA when varying the storage capacity. Initially, the storage capacity of each cluster 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 cluster from 10 to 30 data objects, with each of them having size equal to the average data object size. It is seen in Fig. 23 that the performance of DRA is stable against DBA when varying the surplus storage capacity.

The above is because DBA and DRA do not consider the replication of data objects. It is noteworthy to mention that the network overhead reduction of HPA against DBA and DRA was roughly equal to 35% and 11%, respectively, when the surplus storage capacity of each cluster was equal to 10 data objects. On the other extreme, the network overhead reduction of HPA against DBA and DRA was roughly equal to 49% and 30%, respectively, when the surplus storage capacity of each cluster was equal to 30 data objects.

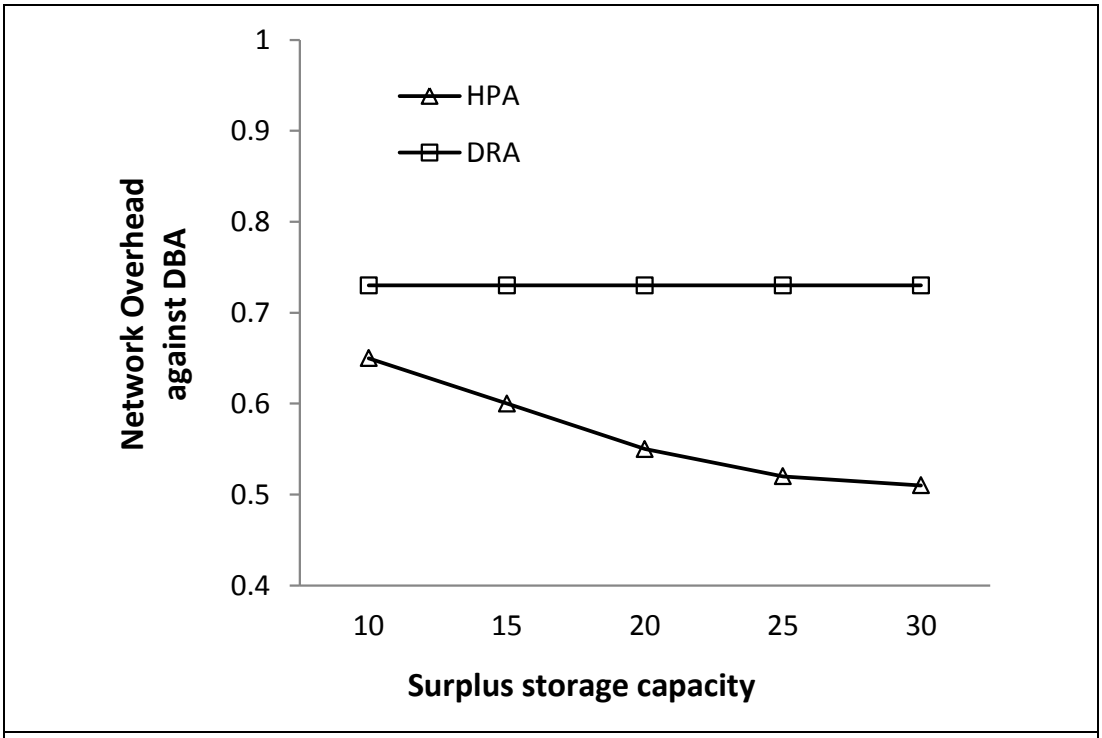

Figure 23. Network overhead when varying surplus storage capacity of each cluster within the system.

6. Related work

The replica placement problem has been researched quite extensively, and a variety of problems definitions have been proposed [15]. In [6] client-replica distance is considered as the optimization target, whereas the primary goal of [31] is load balancing. Read access cost is the focus in [8] and [11], while [12] considers client traffic that includes both read and update requests. In [13] and [14] the authors tackle the problem of minimizing the network overhead during the transition from an old replica assignment scheme to a new one. On the other hand, ref. [27] and [28] tackle the problem of minimizing the time needed for the transition from an old replica assignment scheme to a new one. Other issues taken into account in conjunction with the replica placement problem formulations are server storage capacity [8] and [12], processing capacity [17] and bandwidth [6] to name a few. In this dissertation we have adopted a model similar to [8]. Although our problem definition is related with replicating data objects, it is quite different from the aforementioned problems since our problem combines replication and VM assignment on clusters.

There are also interesting problems that are closely related to our work in the field of virtual machine (VM) placement [4]. The VM placement problem is addressed in [7], whereby the objective is to minimize the network congestion within the system. 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 [30], with the objective being to reduce the hosting cost over time according to both demand and resource price fluctuation. In [16] and [29], the authors target the VM placement problem with their objective being the same with that of our problem. 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] and [26]) is an optimal fully distributed algorithm working also for general-structured application graphs and taking into consideration capacity constraints on nodes.

The problem is also related with the agent placement problem [18], [23], [24], [22], [20], and [21]. In the agent placement problem, the application is consisted of generic and non-generic agents. The generic agents are hosted by any node within the wireless sensor network, while the non-generic agents are hosted only by nodes that fulfill the sensing/actuating requirements of the respective agents. Therefore, a generic agent plays the role of a VM, while a non-generic agent plays the role of data object.

7. Conclusions and outlook

In this thesis we have formulated the joint problem of replicating data objects and assigning the virtual machines onto clusters. 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 clusters. The proposed technique was compared to two state-of-the-art algorithms found in the literature, named DBA [19] and DRA [25]. 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. We also plan to prove that HPA is optimal when there are no storage and capacity constraints on clusters.

References

[1] M. Alicherry, T. V. Lakshman, "Network Aware Resource Allocation in Distributed Clouds," *IEEE Conference on Computer Communications (INFOCOM)*, pp. 963-971, 2012.

[2] M. Al-Fares, A. Loukissas, A. Vahdat, "A Scalable Commodity Data Center Network Architecture," *SIGCOMM*, pp. 63-74, 2008.

[3] O. Goldschmidt, D. S. Hochbaum, "A Polynomial Algorithm for the k-cut Problem for Fixed k," Mathematics of Operations Research, vol. 19, no. 1, pp. 24-37, 1994.

[4] 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*.

[5] B. Heller, et al., "Elastic Tree: Saving Energy in Data Center Networks," *NSDI*, vol. 10, pp. 249-264, 2010.

[6] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," in Proc. *IEEE INFOCOM*, April 2001, pp. 31–40.

[7] J. W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, "Joint VM Placement and Routing for Data Center Traffic Engineering," *IEEE INFOCOM*, pp. 2876-2880, 2012.

[8] J. Kangasharju, J. Roberts, and K. Ross, "Object replication strategies in content distribution networks," *Computer Communications*, vol. 25, no. 4, pp. 367–383, 2002.

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

[10] T. Kosar, M. Balman, E. Yildirim, S. Kulasekaran, B. Ross, "Stork Data Scheduler: Mitigating the Data Bottleneck in E-science," *The Phil. Transactions of the Royal Society*, pp. 3254-3267, 2011.

[11] N. Laoutaris, G. Smaragdakis, A. Bestavros and I. Stavrakakis, "Mistreatrment in Distributed Caching Groups: Causes and Implications," in Proc. *IEEE INFOCOM* 2006, Barcelona, Spain.

[12] T. Loukopoulos and I. Ahmad, "Static and adaptive distributed data replication using genetic algorithms," in *J. Parallel and Distr. Comp. (JPDC)*, Vol. 64(11), pp. 1270-1285, 2004.

[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)*, 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 2007)*, 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*.

[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), pp. 1-9, 2010.

[17] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, "A dynamic object replication and migration protocol for an Internet hosting service," in Proc. *ICDCS'99,* May 1999, pp. 101–113.

[18] I. Rentifis, N. Tziritas, S. Lalis, P. Lampsas, T. Loukopoulos, "Improving Application Availability in Wireless Sensor Networks with Energy-Harvesting Capability," *in 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT),* December 2012.

[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),* pp. 228-237, 2010.

[20] 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, pp. 3143-3161, 2014.

[21] N. Tziritas, S. Lalis, S. U. Khan, T, Loukopoulos, C.-Z. Xu, P. Lampsas, "Distributed Online Algorithms for the Agent Migration Problem in WSNs," *ACM/Springer Mobile Networks and Applications*, vol. 18, no. 5, pp. 622-638, 2013.

[22] N. Tziritas, T. Loukopoulos, S. Lalis and P. Lampsas, "Algorithms for energy-driven agent placement in wireless embedded systems with memory constraints," *Simulation Modelling Practice and Theory (Elsevier),* vol. 19, no. 6, 1445-1464, 2011.

[23] N. Tziritas, P. Lampsas, S. Lalis, T. Loukopoulos, S.U. Khan, C.-Z. Xu, "Introducing Agent Evictions to Improve Application Placement in Wireless Distributed Systems", *in 41st IEEE International Conference on Parallel Processing (ICPP), Pittsburgh,* September 2012.

[24] N. Tziritas, G. Georgakoudis, S. Lalis, T. Paczesny, J. Domaszewicz, P. Lampsas, T. Loukopoulos, "Implementation and Evaluation of Agent Mobility for Wireless Networks of Home Objects," *in 4th International Conference on Sensor Systems and Software (S-CUBE),* June 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, pp. 1690-1704, 2013.

[26] 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.

[27] 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.

[28] 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.

[29] 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),* pp. 449-457, 2013.

[30] 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.
[31] L. Zhuo, C. Wang, and F. Lau, "Load balancing in distributed web server systems with partial document replication," in Proc. *ICPP'02*, August 2002, pp. 305–312.
[32] http://hama.apache.org/
[33] http://hadoop.apache.org/
[34] http://home.web.cern.ch/about/experiments/cms
[35] Network Simulator2 (ns2), http://www.isi.edu/nsnam/ns/
[36] http://swiftstack.com/openstack-swift/architecture/
[37] http://www.genome.gov/
[38] http://www.hpcwire.com/jobs.html
[39] https://www.humanbrainproject.eu/
[40] http://www.sdss.org/

Appendix-A

The most important notations used in this thesis are summarized in the table below.

Table 1

Symbol	Meaning
M	total number of servers
N	total number of objects
S_i	the *i*th server
O_k	the *k*th object
$s(S_i)$	storage capacity of S_i (in data units)
$s(O_k)$	size of O_k (in data units)
l_{ij}	communication cost (per data unit) between S_i and S_j
$X,\ X^{old},\ X^{new}$	The (old, new) replication matrix / placement
P_k	primary server of O_k
N_{ik}^{X}	replicator of O_k nearest to S_i in replica placement X
r_{ik}	read volume arriving at S_i for O_k
T_{ikj}	transfer of O_k from S_i to S_j
D_{ik}	deletion of O_k on S_i
H	schedule of transfer/delete actions
C^{H_u}	cost of the *u*th action of schedule *H*
$I_H^{X,X'}$	cost of a valid schedule *H* that leads from *X* to X'