GreenDataNet

D3.8 – Aggregated Energy Management System

Draft
EPFL
Rev 2.0
EPFL: Pablo Garcia, Ali Pahlevan, David Atienza

TABLE OF CONTENTS

REVISION SHEET

Revision Number	Date	Brief summary of changes
Rev 0.1	21/12/2015	First version
Rev 0.2	14/01/2016	Adding content
Rev 1.0	18/01/2016	Completed content
Rev 2.0	16/03/2016	Updated conclusions based on consortium feedback

KEY REFERENCES AND SUPPORTING DOCUMENTATIONS

[1] A. Amokrane and et al., “Greenhead: virtual data center embedding across distributed infrastructures,” IEEE TCC, 2013.

[2] F. Researc, “The future of data center wide-area networking,” 2010.

[3] J. Kim and et al., “Correlation-aware virtual machine allocation for energy-efficient datacenters,” in DATE, 2013.

[4] M. Pedram and et al., “Power and performance modeling in a virtualized server system,” in ICPPW, 2010.

[5] J. Kim and et al., “Free cooling-aware dynamic power management for green datacenters,” in HPCS, 2012.

[6] C. Bergonzini and et al., “Comparison of energy intake prediction algorithms for systems powered by photovoltaic harvesters,” Microelectronics Journal, 2010.

[7] D. Dias and L. Costa, “Online traffic-aware virtual machine placement in data center networks,” in GIIS, 2012.

[8] L. Gu and et al., “Joint optimization of vm placement and request distribution for electricity cost cut in geo-distributed data centers,” in ICNC, 2015.

[9] O. Biran and et al., “A stable network-aware vm placement for cloud systems,” in CCGRID, 2012.

2. INTRODUCTION

2.1 DOCUMENT PURPOSE

This deliverable describes the Aggregated Energy Management System, that is directly linked to Task 3.3.1: Communication Strategies Between Aggregator and Data Centres, where the interface between the SEMS (Smart Energy management System, See D3.7) of each data centre and the AEM (Aggregated Energy management System), in charge of the optimization strategies, are described.

Task 3.3.3: Virtualisation of IT Tasks and Optimised Dispatching and Task 3.4: Design and Implementation of a Distributed Framework for State Estimation in the Power Grid are also related, since they both perform multi-level optimizations driven from the networked data centres and that require as input parameters on a global level the IT loads for the overall network of data centres, and potential control signals coming from a smart grid operator, plus the Individual input parameters from each data centre (electricity consumption and the PV production forecasting, as well as the battery state of charge and the local electricity pricing profiles).

2.2 DEFINITION, ACRONYMS AND ABBREVATIONS

2.2.1 KEY ACRONYMS AND ABBREVATIONS

AEMS	Aggregated Energy management System
BER	Bit Error Rates
CPU	Central Processing Unit
DC	Data Centre
DoD	Depth of Discharge
ePDU	Rack Power Distribution Unit
HW	Hardware
IT	Information Technology
PDU	Power Distribution Unit
PMSM	Power Monitor System and Management
PUE	Power Usage Effectiveness
PSU	Power Supply Unit
PV	PhotoVoltaic
UPS	Uninterruptible Power Supply
SEMS	Smart Energy management System
SW	Software
QoS	Quality of Service
VM	Virtual Machine

2.3 DOCUMENT OVERVIEW

Section 3 of this deliverable explains the role of The Aggregator; the orchestrator that implements the necessary communication strategies to make the Smart Energy management System discuss with the higher lever manager (AEMS) to optimize a cluster of Data Centers. Then, in Section 4 we describe the network model that allows the system to interchange information. Next, Section 5 presents a test case of geo-distributed DC optimization using a global optimization algorithm, with the corresponding results and analysis in Sections 6 and 7.

3. THE AGGREGATOR

The Aggregated Energy Management System describes a layer that receives all the information from the DCs. More exactly, it collects statistics from:

- the microprocessor
- the Operating System
- the Hypervisor
- the smartPDUs and power units
- the additional HW instrumentation (PMSM sensors, for instance)

Then, thanks to all these information, it can efficiently manage the temperature, performance, and power of the system. The main constraint is to fully respect the IT load. Following objectives are to minimise the energy consumption of the network of data centres, maximise the renewable energy share, and facilitate the integration within smart grids.

Initially, a special, dedicated device, “The Aggregator”, was conceived to realize such a task, interfacing all the components of the system. However, due to the final specifications, The Aggregator was no longer needed, since its functionality has been integrated in the Rack Controller and the algorithm that is running on it (see deliverables D2.4 and D3.7). More exactly, the mission of The Aggregator was to provide a compatibility layer to allow for integration of different components from different manufacturers. However, during the development of the project, we have always stuck to the same providers: Eaton (UPS, ePDUs), Credit Suisse (Servers), CEA (PVs, chargers) and Nissan (Batteries); thus, making unnecessary this extra layer.

Our aggregator (Rack Controller) implements the necessary communication strategies to make the Smart Energy management System (SEMS, that optimize the consumption of one DC) discuss with the higher lever manager (AEMS) which is able to optimize a smart grid or a cluster of Data Centers. The aggregator is the orchestrator at the very high level, and it communicates with the DC components using the network; therefore, the first section of this deliverable describes the network and latency models used. Then, to prove the advantages of using a well designed data aggregator that scales correctly, a case study will be presented where we optimize a geo-distributed DC using the described infrastructure.

4. NETWORK AND LATENCY MODEL

The network communications inside geo-distributed DCs play a very important role [1], since not only the data has to travel from server to server, also the control commands and information must be shared globally. Therefore, all the aggregated information must travel through both local and global links, incurring in some delays and a certain latency. For this reason, we have developed the Network and Latency Model that is described next.

In order to model the communications accurately, our algorithm considers intra-DC local links with bandwidth (BL) (to access the network-attached storage), and inter-DC connections, modeled as a full mesh backbone network topology with bandwidth (Bbb). The global links are modeled in the presence of bit error rates (BERs) and their probabilities (PBER) associated to the data transmission, the speed of light, and distance between DCs. To compute the total latency for both migrating a set of VMs (according to VMs size) at time slot T and data communication during the time interval of (T;T +1) from multiple DCs to a specific DC, we take into account two parts:

1) local and global latency for the ith source DC, i.e. Li l and $L^{i,j}_g$ respectively, to transmit information through the local and global networks to the jth destination DC
2) local latency for the jth destination DC (L^j_l) to transmit data collected from other DCs to its storage.

Equation 1 represents the total (worst-case) latency for the jth destination DC (L^j_t) as the summation of the maximum latency between source DCs for transmitting the corresponding data through their dedicated local and global links, and local latency inside destination DC. N_{DC} is the total number of DCs.

(1) $L^j_t = \text{maxi}(L^i_l + L^i_l + L^{i,j}_g) + L^j_l \qquad i = 1 \text{ to } N_{DC} \text{ and } i \neq j$

Local latency of the ith source DC is dependent on the volume of data ($Vol^{i,j}$) ready to be transferred to the jth destination DC and its local bandwidth (B^i_L). Therefore, each source DC local latency is calculated as:

(2) $L^i_l = (Vol^{i,j})=B^i_L$

The local latency of the jth destination DC is related to the total volume of data received from the multiple source DCs and its local bandwidth (B^j_L), computed as:

(3) $L^j_l = \Sigma^{NDC}_{i=1,i\neq j} Vol^{i,j} / B^j_L$

The global latency includes propagation latency as a primary source and data latency with respect to the volume of data being transmitted. Propagation latency is a function of how long the data takes to travel at the speed of light (S_l) from source to destination (distance: $Dist_{i,j}$). Data latency ($L^{i,j}_e$) is a function of the effective bandwidth (Be(t)) and the (BER(t)) (corrupted data must be resent). Hence, the global latency is calculated as:

(4) $L^{i,j}_g = Dist_{i,j} / S_l + L^{i,j}_e$

To calculate the data latency (L_e) in the presence of transmission errors, first we calculate the effective bandwidth and, then, we fragment the transmission into the necessary number of time steps. Figure 4.1 (Algorithm 1) describes this process analytically.

1: **while** true **do**
2: $B_e(t) = (1 - BER(t)) \cdot B_{bb}, \quad BER(t) \propto P_{BER(t)}$
3: **if** $Vol^{i,j} \leq B_e(t)$ **then**
4: $L_e = L_e + Vol^{i,j}/B_e(t)$
5: *Break*
6: **else**
7: $Vol^{i,j} = Vol^{i,j} - B_e(t)$
8: $L_e = L_e + 1$
9: **end if**
10: **end while**

Figure 4.1 – Algorithm 1 - Global Data Latency (L_e) w.r.t BER

5. OPTIMIZATION ALGORITHM FOR GEODISTRIBUTED DCS

While optimal VM placement placement in geo-distributed DCs is a NP-complete problem, we propose a two-phase algorithm with low computational overhead that can be applied in real-time. It consists of VMs clustering for DCs (global dispatching controller), and allocating clusters to the servers (local controller). At each time slot T, first the global controller receives the VMs' loads from the previous time interval [T–1,T), data communications, renewable forecast, available battery energy and grid price from each DC. Then, we cluster the VMs (available VMs in the system and newly arrived), for each DC. After clustering, at local level, distributed in each DC, the VMs are allocated to the minimal number of servers. During the time interval of [T,T+1), the local green controllers in each DC compensate the difference between real and forecasted load and renewable information.

5.1 GLOBAL PHASE - VMS CLUSTERING

We split this phase into three different steps. First, at time slot T, all the VMs available in the system are represented as points in a two dimensional plane (2D plane). Based on the data and CPU-load correlation properties, as highly data-correlated VMs should be clustered together while highly CPU-load correlated VMs should be placed apart, a function is defined to calculate attraction and repulsion forces between each two VMs. Equation 5 calculates the force from ith to jth VM ($F^{i,j}_t$) as a function of attraction force ($F^{i,j}_a$) based on the data correlation ($Corr^{i,j}_{data}$) normalized as [-1,0), and repulsion force ($F^{i,j}_r$) based on the CPU-load correlation ($Corr^{i,j}_{cpu}$) normalized as (0;1]. The attraction force from i^{th} to j^{th} VM is different from j^{th} to i^{th} VM due to the consideration of bidirectional data correlation and calculated as amount of data two VMs exchange. The repulsion force is computed as a worst-case peak CPU utilization when the peaks of two VMs coincide during the last time slot. α denotes a weighting factor for energy and performance trade-off calculation.

$$(5) \qquad \begin{cases} F_a^{i,j} = Corr_{data}^{i,j} \\ F_r^{i,j} = Corr_{cpu}^{i,j} \end{cases} \Rightarrow \; F_t^{i,j} = \alpha \; . \; F_a^{i,j} + (1-\alpha) \; . \; F_r^{i,j}$$

Initially, at time slot 0, all the points are distributed in the 2D plane. Then, the resultant forces in the X (F^i_x), and Y (F^i_y) directions are calculated amongst points (θj,i is the angle) and, as a result, the

points are remapped in the 2D plane with new coordinates ($Loc^i_x(k)$;$Loc^i_y(k)$) at each iteration k as follows:

$$(6)\quad \begin{cases} F_x^i = \sum_{j=1, j\neq i}^{N_{vm}} F_t^{j,i}.\cos(\theta_{j,i}) \\ F_y^i = \sum_{j=1, j\neq i}^{N_{vm}} F_t^{j,i}.\sin(\theta_{j,i}) \\ Loc_x^i(k) = Loc_x^i(k-1) \ + \ 0.5 \ . \ F_x^i(k) \ . \ t^2 \\ Loc_y^i(k) = Loc_y^i(k-1) \ + \ 0.5 \ . \ F_y^i(k) \ . \ t^2 \end{cases}$$

where Nvm and t denote the number of VMs (points) available in the system and time period of displacement, respectively. The process is iterated until the cost function ($Cost^{AR}$) of the current iteration k, expressed as Eq. 7, becomes worse than that in the previous one k-1. We also fix a maximum number of iterations to avoid a convergence time overhead.

$$(7)\quad Cost_k^{AR} = \sum_{i=1}^{N_{vm}} \sum_{j=1, j\neq i}^{N_{vm}} F_t^{i,j}.(d_k^{i,j} - d_{k-1}^{i,j})$$

where $d^{i,j}_k$ depicts the distance between i^{th} and j^{th} points at iteration k. This function demonstrates if there is either an attraction force between each pair of points ($F^{i,j}_t < 0$), and they are attracted to each other ($d^{i,j}_k – d^{i,j}_k - 1 < 0$), or a repulsion force ($F^{i,j}_t > 0$), and they separate away. The final location of all the VMs becomes the initial position for the next time slot. In the second step, we first define a capacity cap (in Joules) per each DC (cluster) to minimize the operational cost, computed according to the available battery energy, renewable energy forecast, grid price and DCs power consumed during the last previous time slot; i.e., last-value predictor.

Then, we utilize a modified version of the k-means algorithm to cluster VMs with respect to each cluster capacity cap, VMs load, and the distance between two VMs obtained from the repulsion and attraction phase in the 2D plane. In the modified k-means, the initial centroid of each cluster is calculated based on the last position of points available in that cluster in the previous time slot. In this step, we do not consider network latency. At the last step, we revise the modified k-means output to meet the hard time constraint for migrating VMs across DCs based on their size as described in Algorithm 2 (Figure 5.1). The output of the modified k-means creates two queues per cluster (DC): outgoing and incoming. The first one contains the candidates to be migrated outside, to another DC, sorted in descending order according to their distances from the corresponding cluster's centroid (Qout). The second one contains the candidates to be migrated to this DC sorted in ascending order (Qin).

The algorithm first selects one DC (ith DC) and checks if its previous load (Ri) is less than its capacity cap (Cap_i). Then, it selects the first VM from the head of the incoming queue of the cluster ($Head(Q^i_{in})$). We migrate this VM if the latency allows; otherwise, we erase it from the queue and select the next VM. We repeat and update the DC's load until there is either no VM to accept or the load of the DC becomes more than the cap (lines 5-12). In this later case (lines 13-24), we select the VM from the head of the outgoing queue of the current cluster (Head(Qi out)) which has the maximum distance to the centroid. If this VM can be migrated, we check the current load of the destination cluster and repeat this process there. Otherwise, we select the next one in the cluster. This algorithm iterates until violating the latency constraint for all DCs or there is no action to do. Unallocated VMs that have been available in the system will stay in their previous DC, and unallocated new VMs are assigned to the DCs determined from the modified k-means step without

the consideration of the network latency constraint. In this case, we have tried to find the best solution for migrating the appropriate VMs when the number of migrations is bounded. We also prevent network bottlenecks made by one DC when the other DCs need to migrate their VMs to the same destination DC.

Input: Outgoing and incoming queues
Output: VMs migration actions
1: $Q^i_{out} \leftarrow$ Sort i^{th} DC outgoing queue based on VMs distances from its centroid (Descending order)
2: $Q^i_{in} \leftarrow$ Sort i^{th} DC incoming queue based on VMs distances from its centroid (Ascending order)
3: $i \leftarrow 1$ Initial DC
4: **while** (Q_{in} and Q_{out} are not $NULL$ for all DCs) & (Latency constraint is not violated for all connections) **do**
5: **if** $R_i < Cap_i$ **then**
6: VM = Head (Q^i_{in})
7: $j \leftarrow$ Current DC of VM
8: **if** $L^i_t <$ Latency constraint **then**
9: Migrate VM from j^{th} DC to i^{th} DC
10: Update i^{th} and j^{th} DCs' load (R_i and R_j)
11: **end if**
12: Erase VM from Q^i_{in} and Q^j_{out}
13: **else if** $R_i \geq Cap_i$ **then**
14: VM = Head (Q^i_{out})
15: $j \leftarrow$ Destination DC of VM
16: **if** $L^j_t <$ Latency constraint **then**
17: Migrate VM from i^{th} DC to j^{th} DC
18: Update i^{th} and j^{th} DCs' load (R_i *and* R_j)
19: Erase VM from Q^i_{out} and Q^j_{in}
20: $i \leftarrow j$ Move to destination DC
21: **else**
22: Erase VM from Q^i_{out} and Q^j_{in}
23: **end if**
24: **end if**
25: **end while**

Figure 5.1 – Algorithm 2 - Migration Step - Modified K-means Output Revision

5.2 LOCAL PHASE - VMS ALLOCATION

At the local phase, the VMs of each cluster are allocated to servers of their corresponding DC, and the optimal frequency for each server is computed. We use only CPU-load correlation to allocate VMs to the minimum number of servers, since data correlation (and migrations) mainly contribute to inter-DC network bottlenecks [2], [1]. Hence, we base our implementation on the best state-of-the-art algorithm [3] for VMs allocation.

5.3 THE GREEN CONTROLLER

The proposed VM placement algorithm reduces the dependency on grid energy based on the load and renewable forecast. Therefore, we require a low-complexity green controller to compensate the difference between real and forecast information with respect to the current electricity price of DCs. After allocating all the VMs to servers at time slot T, the green controller

inside each DC manages the energy sources during the time interval of [T;T +1) based on the real renewable energy and DC energy consumption. When the available renewable energy is more than the DC energy consumption, we use this free energy for the DC and the excess energy is stored in the battery bank. Otherwise, during the high price period, we use the whole renewable energy for the DC's load and, for the remaining load, we discharge the battery considering its depth of discharge (DoD). During the low price periods, we charge the battery by grid energy and we do not use it for the DC.

6. EXPERIMENTS

We consider three different DCs located in Europe: Lisbon (DC1), Zurich (DC2) and Helsinki (DC3), along with their distances (for the network model), time zone and two-level real electricity price scenario. Each DC contains 10 rooms and, each room, has 150, 100 and 50 servers for DC1, DC2 and DC3, respectively. Table 6.1 summarizes the number of servers, PV module size and lithium-ion battery capacity (with 50% of DoD, keeping the remaining capacity in case of outage) per DC. We target an Intel Xeon E5410 server consisting of 8 cores and two frequency levels (2.0GHz and 2.3GHz), and use the power model in [4]. For cooling power consumption, we use a time-varying PUE model, as in [5]. The DCs are connected through 100 Gb/s full duplex peer-to-peer optical fiber links, and the intranet uses 10 Gb/s full-duplex links. Global links experience a BER that is chosen randomly from the following distribution: 54% probability of 10^{-6}, 20% of 10^{-5}, 15% of 10^{-4}, 10% of 10^{-3}, and 1% of 10^{-2}.

In order to simulate a realistic scenario, DC VMs and energy demand, we sampled the VMs' utilization of a real DC every 5 seconds for one day, and extended it to 7 days by adding statistical variance with the same mean as the original traces. For renewable forecast, we implemented the algorithm in [6]. Arrival and life-time of each VM, given in time slots, are generated by poisson and exponential distributions, respectively. Data correlation between each pair of VMs is generated by a log-normal distribution with the mean of 10 MB and uniform variance selection in the range of [1, 4] [7]. For migration, the size of the VMs are in the range of 2, 4, and 8 GB according to the distribution of 60%, 30%, and 10%. Finally, the global and local controllers are invoked every one hour, and the green online controller in each DC is invoked every 5 seconds. We also take into account a hard time constrain for migrating the VMs across DCs through the network which guarantees 98% quality of service (QoS).

DC	Number of Servers	PV Capacity (KWp)	Battery Capacity (KWh)
*DC*1	1500	150	960
*DC*2	1000	100	720
*DC*3	500	50	480

Table 6.1 - DCs number of servers and energy sources specification.

7. RESULTS

We compared our algorithm against three state-of-the-art approaches that are the best in their class to optimize operational costs, energy consumption and performance (response time), respectively:

- Cost-aware approach (Pri-aware) [8].
- Energy-aware VM allocation (Ener-aware) [3].
- Network-aware VM placement (Net-aware) [9].
- Proposed: the proposed multi-objective VM placement.

All the mentioned methods are used jointly with the same local green controller to manage battery and renewable energy.

Operational cost

Figure 7.1 shows the operational cost normalized by the worst-case value among the mentioned methods in the time horizon of one week: 55, 25 and 35% cost savings for the proposed method compared to Ener-aware, Pri-aware and Net-aware, respectively. Proposed clusters the VMs by specifying a load cap for different DCs based on the grid price and available renewable and battery energy. It outperforms the other algorithms when a local energy-aware VM allocation is utilized to further reduce DCs' dependency on grid energy. Differently, Ener-aware uses CPU-load correlation to reduce energy consumption and cost in each DC locally but, globally, it cannot efficiently cluster and dispatch VMs for right DCs based on available renewable energy, battery status and grid price. In Pri-aware, the VMs are packed and placed onto DCs and servers with the lowest current grid price, but it neglects to maximize free energies usage. Finally, the Net-aware approach provides load balancing across DCs which in turn leads to better exploiting free energies (renewable and battery) compared to Ener-aware and Pri-aware. However, this algorithm does not consider the electricity price diversities and neglects to utilize an energy-efficient management to reduce its dependency on the grid.

Energy consumption

Figure 7.2 shows the hourly energy consumed by the DCs for one week. The total energy consumption is 57, 55, 65 and 67 GJ for the Proposed, Ener-aware, Pri-aware and Net-aware methods, respectively. The results show 12 and 15% energy improvements for our proposed algorithm compared to Pri-aware and Net-aware due to the consideration of the CPU-load correlation between VMs, that places highly CPU-load correlated VMs apart, in different DCs and servers. This favors consolidation and leads to power savings by lowering the number of active severs and their operating frequency. On the other hand, the Ener-aware approach first uses the FFD clustering heuristic, placing VMs into the first DC in which its load capacity fits, and then packs the VMs into the minimal number of active servers based on the CPU-load correlation. Hence, the DC local controller finds a better mapping of VMs to servers when most of the VMs are in the same DC. Our algorithm, however, tries to find the best VMs clusters per each DC based on the CPU-load and data correlations and determined DCs' capacity cap. Although these correlations indicate opposed goals for energy and performance, Ener-aware only results in 3% energy improvement compared to our multi-objective algorithm, while significantly degrading operational costs and performance (shown in the next section).

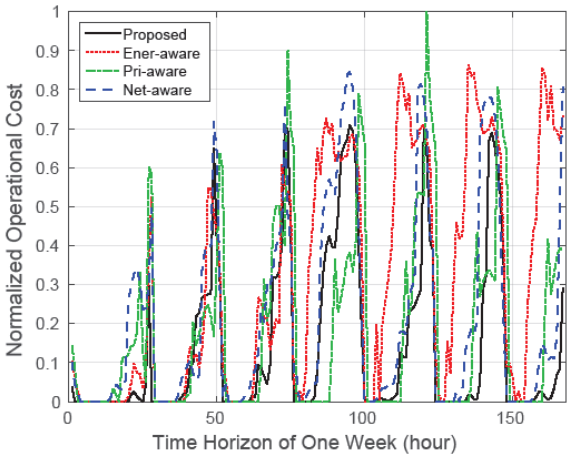

Figure 7.1 - Normalized operational cost for time horizon of one week.

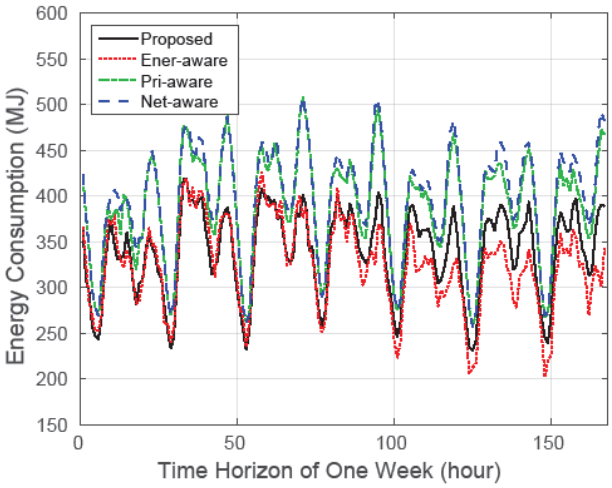

Figure 7.2 - Energy consumed by DCs for time horizon of one week.

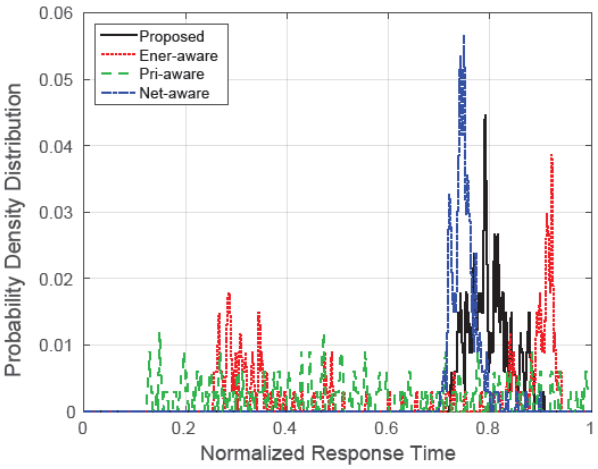

Figure 7.3 - Probability distribution of normalized response time in one week.

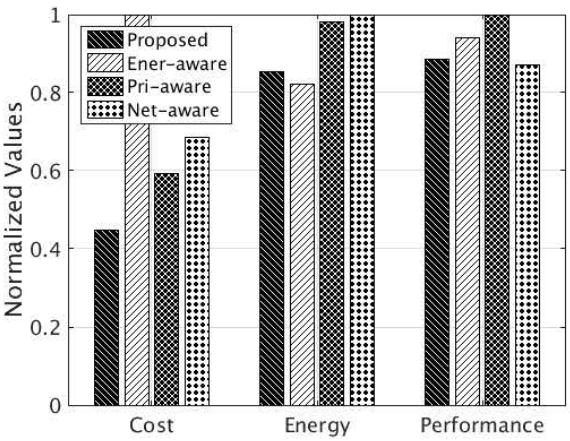

Figure 7.4 - Total cost, energy and performance.

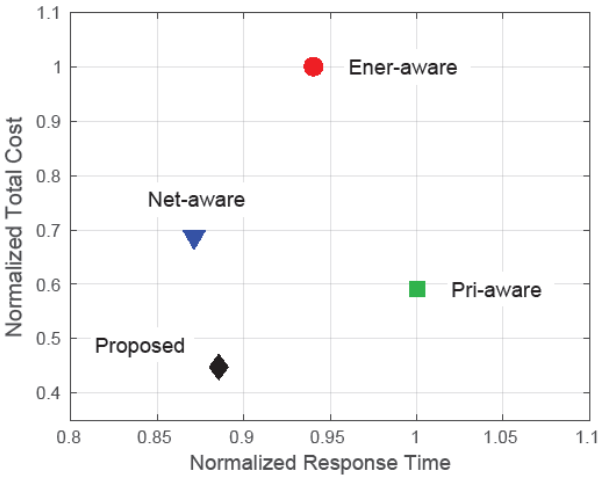

Figure 7.5 - Cost-Performance trade-off.

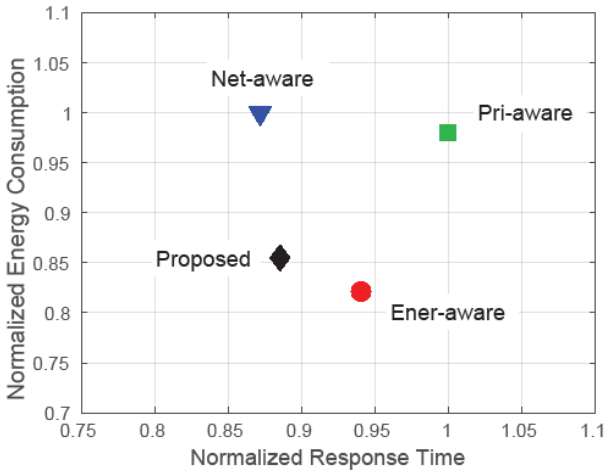

Figure 7.2 - Energy-Performance trade-off.

Performance

In this context, performance is defined as response time due to data communication between VMs through the network. Figure 7.3 shows the probability density distribution of the response time in one week. Note that the response time results are normalized with respect to the worst-case value among the methods. As a result, Proposed and Net-aware encompass a range of response time with higher average and lower variance compared to Ener-aware and Pri-aware methods. The goal of Net-aware is to balance the network across DCs, which in turn leads to better worst-case and higher average response time (both for times of high and low data demands between VMs). However, when compared to Proposed, Net-aware only achieves 2% performance improvement. Ener-aware and Pri-aware tend to place the VMs on a lower number of DCs, which leads to unbalanced network traffic with bigger fluctuations and, accordingly, lower average response time. However, as DCs providers consider worst-case response time to guarantee QoS, the proposed algorithm results in up to 12% performance improvement compared to state-of-the-art approaches.

Trade-offs discussion

The experimental results confirm that, by having a holistic approach, we can obtain better trade-offs in the problem of VM placement. Figures 7.4, 7.5 and 7.6 summarize the benefits of Proposed: In the first place, Fig. 7.4 depicts the totals, showing up to 55, 15 and 12% improvements for operational cost, energy consumption and performance, respectively. Then, Fig. 7.5 shows the cost-performance trade-off, with Proposed providing 25 and 12% improvements for cost and response time, respectively, compared to Pri-aware. In comparison with Net-aware, it achieves 35% cost savings while it leads to only 2% performance degradation. Finally, Fig. 7.6 exhibits the energy-performance trade-off: our algorithm results in 6% performance improvement with a 3% energy overhead compared to Ener-aware; and it provides 15% energy savings and 2% performance degradation compared to Net-aware.

8. CONCLUSIONS

This deliverable is focused on the aggregator, the orchestrator that makes possible the tasks of control and optimization of geo-distributed DCs. We described the network model that intercommunicates the different elements for information interchange and we proposed a novel method to tackle the challenges of operational cost optimization and energy-performance trade-off on resource-constrained green geo-distributed DCs.

We introduced the two-phase multi-objective VM placement algorithm along with a dynamic migration technique that exploit the holistic knowledge of VMs characteristics. The first phase, i.e. global controller, clusters VMs for each DC considering time-varying VMs CPU-load and data correlations and the status of DC energy sources. The second phase, i.e. local controller, allocates the VMs of each DC cluster to servers exploiting CPU-load correlation.

Finally, experimental results using a complete simulation framework for geo-distributed datacenters developed in GreenDataNet showed that, using the proposed algorithm and infrastructure, up to 55, 15 and 12% improvements can be obtained for operational cost, energy consumption and performance, respectively, compared to state-of-the-art approaches. It is therefore, better to aggregate all the data coming from all the elements of the DCs in order to be take more optimized, global, decisions.