++++++++++ Algorithm 1: EViMA

Input : Workflow, set of VMs and set of VM types (VMheC, VMmec, VMlec)

Output:

wtReadyPool = (wt1, wt2, tw3,…wtn)

while wtReadyPool # do

Compute EFT ofeach tasks

Group Tasks => HCT, MCT, LCT

Group VMs => VMhec, VMmеc, VMlec

foreach wt in wtReadyPool do

if wt, HCT and wt, LCT then

apply algorithm 2 to execute HCT

apply algorithm 3 to execute LCT

else

if EFT ofwt, = Dl ofwt, then

wt, +> VMmec

Update wtReadyPool

Else

if EFT of wt, ST then

apply algorithm 5

else

apply algorithm 4 to save mood the idle VM

Update wtReadyPool

end

end

end

end

end

++++++++++++

++++++Algorithm 2: HiCTSA

Input : • Workflow DAG, G=(W,T) where W= (wt1 ,wt2 ,wt3 wtn

• A set of cloud resources with different CPU configuration and different pricing

Output:

Check tasks criticality in wtReadyPool

Compare EFT of ww1 on VM1 to its Dl

if EFT ofwt1 > Dl ofwt, then

Call wt1 HCT

 Map all HCT to VMheC 6 else

 if DLwt==DLwt,

deploy VMheс?

If EFT ofwt, = Dl ofwt, then

Call the task MCT

Update wtReadyPool

while wtReadyPool @do

|Repeat step 3 to 8

End

if wtReadyPool = ф

else

end

++++++++++++++++++++++++++++++

Algorithm 3: LoCTSA

Compare EFT of wt1 on VM1 to its DI

 if EFT ofwt, < Dl ofwt; then

 Call wt1 LCT

Map all LCT to VMlec

 Else

if EFT ofwt, = Dl ofwt,, then

move wt; into MCT queue

Update wtReadyPool

while wtReadyPool do

Repeat step 2 to 8

end

ifwtReadyPool = ф

else

end

++++++++++++++++++++++++++++++

Algorithm 4: VM-PRA

Output:

VMPool = (VMнbс, VMmвс, VMLbc,…VMk)

 foreach VM in the VMPoot do

Check VMs status (busy or idle)

if VMhec is idle then |Put VMhec on save mood else

if VMmec and VMlec are idle then

Put VMmec and VMlec on save mood

Update VMpool

while VMPoot #фdo

|Repeat step 2 to 8

end

if VMPool = ф

else

end

end

++++++++++++++++++++++++++++++++

Algorithm 5: STiHaM

Input : • Most workflow tasks have slack times Output:

WtReadPool = (wt1, wt2, tw3,wtn)

 VMPool = (VM,, VMs, VM, …VMk)

 foreach VM in the VMPool do

Generate the EFT and ST of all tasks

Harvest the slack time of each VM

Store the slack times of each VM in STHub

Prompt the scheduler when EFT of wtij = ST of VMi

Allocate wtij to utilize the slack time

Update VMpool

while VMPoot #odo

| Repeat step 4 to 8

end

if VMPool = ф

end

++++++++++++++++++++++++++++++++++++++

Amélioration

Ajouter l'algorithme DVFS

# Fonction pour mesurer la charge de travail

def mesure\_charge\_travail():

# Code pour mesurer la charge de travail

# Peut être basé sur l'utilisation du CPU, la charge système, etc.

# Retourne une valeur représentant la charge de travail

# Fonction pour prédire la charge de travail future

def predire\_charge\_travail\_future():

# Code pour prédire la charge de travail future

# Peut être basé sur des techniques de prédiction, des données historiques, etc.

# Retourne une valeur représentant la charge de travail future

# Fonction pour sélectionner la fréquence et la tension optimales

def selectionner\_frequence\_tension\_optimales(charge\_travail):

# Code pour sélectionner la fréquence et la tension optimales en fonction de la charge de travail

# Peut être basé sur des seuils prédéfinis, des tables de correspondance, des algorithmes d'optimisation, etc.

# Retourne la fréquence et la tension optimales

# Fonction pour ajuster la fréquence et la tension du processeur

def ajuster\_frequence\_tension(frequence, tension):

# Code pour ajuster la fréquence et la tension du processeur

# Peut nécessiter des autorisations spéciales pour accéder aux paramètres de fréquence et de tension du processeur

# Peut varier en fonction du système d'exploitation et de l'architecture matérielle

# Utiliser avec précaution pour éviter d'endommager le matériel

# Boucle principale pour l'algorithme DVFS

while True:

# Mesurer la charge de travail actuelle

charge\_actuelle = mesure\_charge\_travail()

# Prédire la charge de travail future

charge\_future = predire\_charge\_travail\_future()

# Sélectionner la fréquence et la tension optimales

frequence\_optimale, tension\_optimale = selectionner\_frequence\_tension\_optimales(charge\_actuelle)

# Ajuster la fréquence et la tension du processeur

ajuster\_frequence\_tension(frequence\_optimale, tension\_optimale)

# Attendre un certain temps avant de recommencer la boucle

# Peut être ajusté en fonction des besoins de l'application

time.sleep(1)