The architecture of Kaligreen V2: A middleware aware of hardware opportunities to save energy

UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

Author

Hernán Humberto Álvarez Valera

Supervisors

Marc Dalmau, Philippe Roose
Christina Herzog

1. Motivation

Shocking data: the reason to worry about energy...

*...The **data center sector** was estimated to have consumed about **61 billion kilowatt/hours** (kWh) in 2006 (**1.5 percent** of total U.S. electricity consumption) for a total electricity cost of about **$4.5 billion (2006 dollars).** The electricity use of the nation's servers and data centers in 2006 was more than double the electricity that was estimated to have been consumed for this purpose **in 2000. .***

Electricity demand increases from about 29 billion kWh in 2000 to nearly 73 billion kWh by 2020

...Consider CO2 emission ...

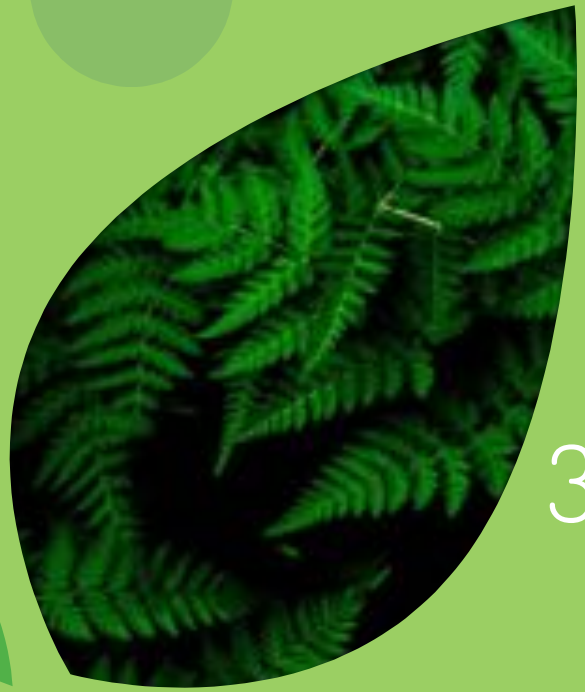
2. how software can save energy ?

Let’s talk about load balancing...

Cloud: Strategies and policies to process environmental data

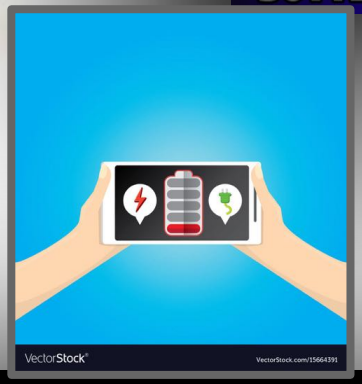
Data Center:Load balancing strategie: From the conception of a distributed application to its deployment.

Host: Load balancing algorithms and correct use of programming tools



3. What is our Problem?

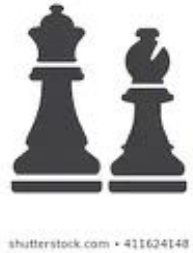

WHAT IS BOTTLENECK?
COMMON
BOTTLENECK HARDWARE!

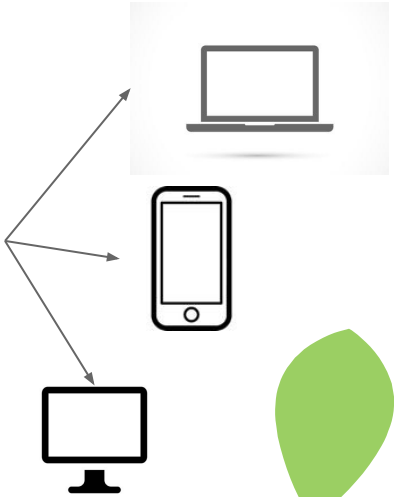

VectorStock


With i7-6820HK
Overclocking
4.0GHz+
+14%
With i7-6820HK
P15000+
+P1000~
With i7-6700HQ
3.5GHz
With i7-6700HQ
P13800
Over clock
3.5 GHz
4.0 GHz

KALIMUCHO

> *There is no decentralized way to deploy and manage an application based on microservices through user devices in order so save energy*

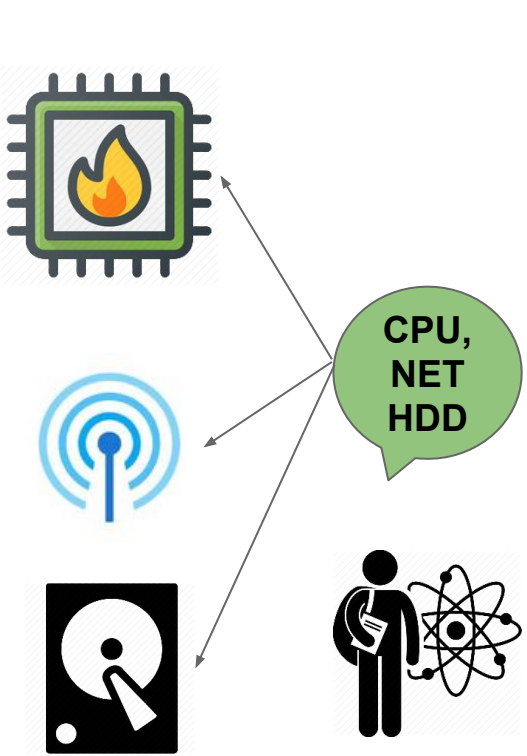


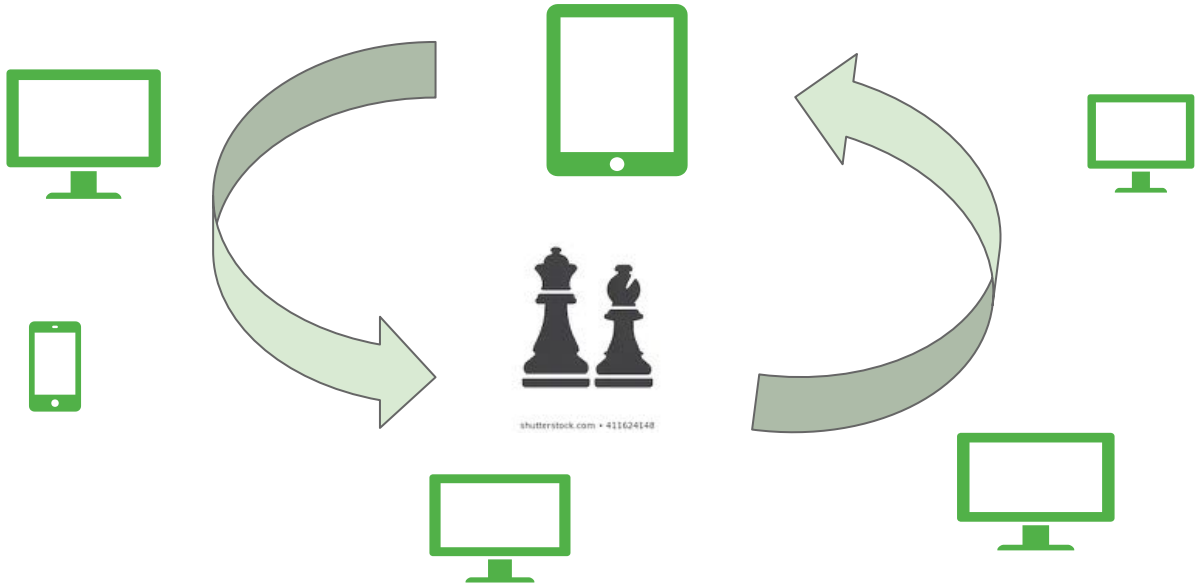


4. So, What is Kaligreen?

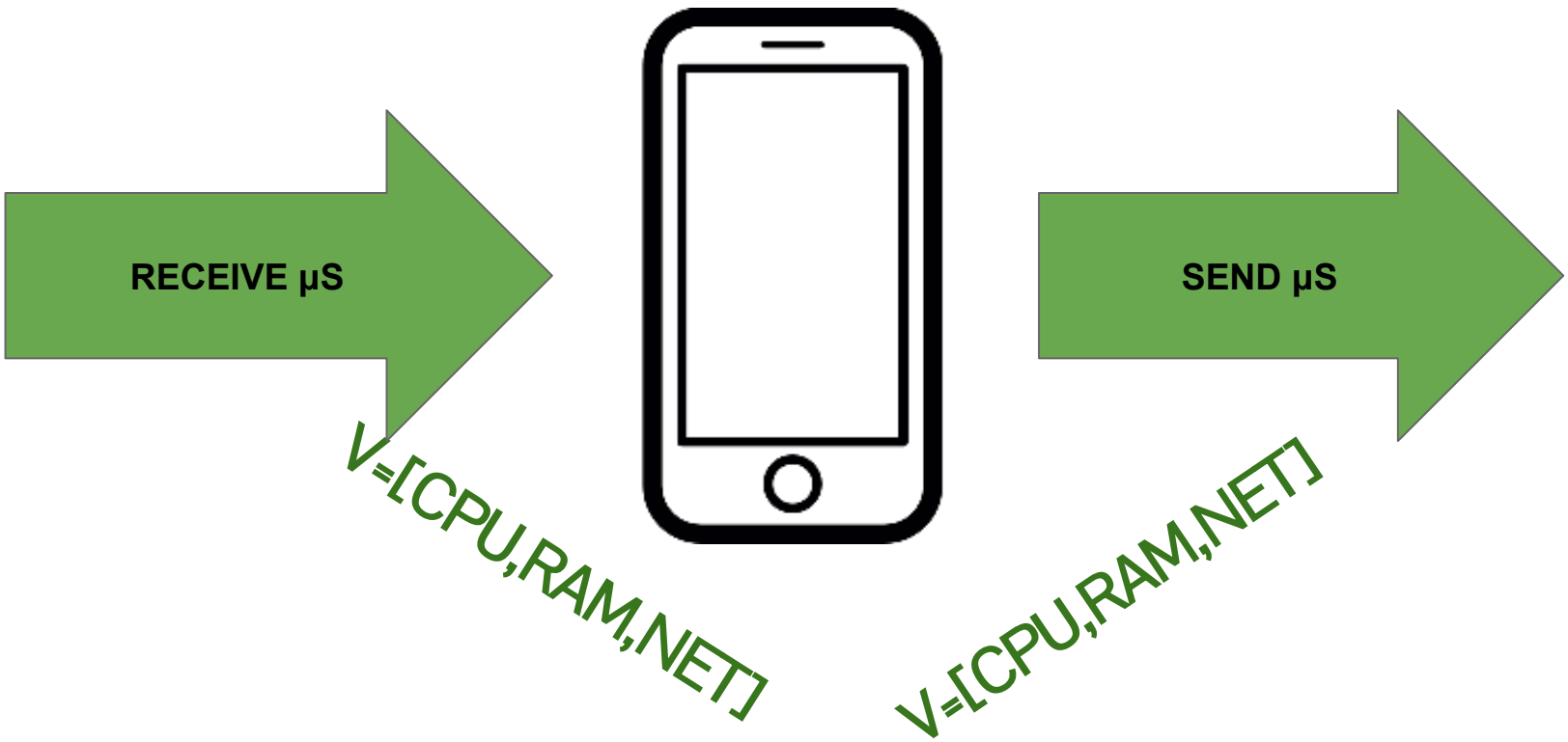
It is an autonomic extension of Kalimucho middleware capable of running distributed applications based on microservices hosted on user devices.

Can move/remove/duplicate/change microservices to save energy.





5. Kaligreen V1


RECEIVE µS
SEND µS
V=[CPU,RAM,NET]
V=[CPU,RAM,NET]

C100 – R50 – N50- B4

C80– R50 – N50

C10 – R30 – N10

C15 – R50 – N50

C100 – R90 – N50

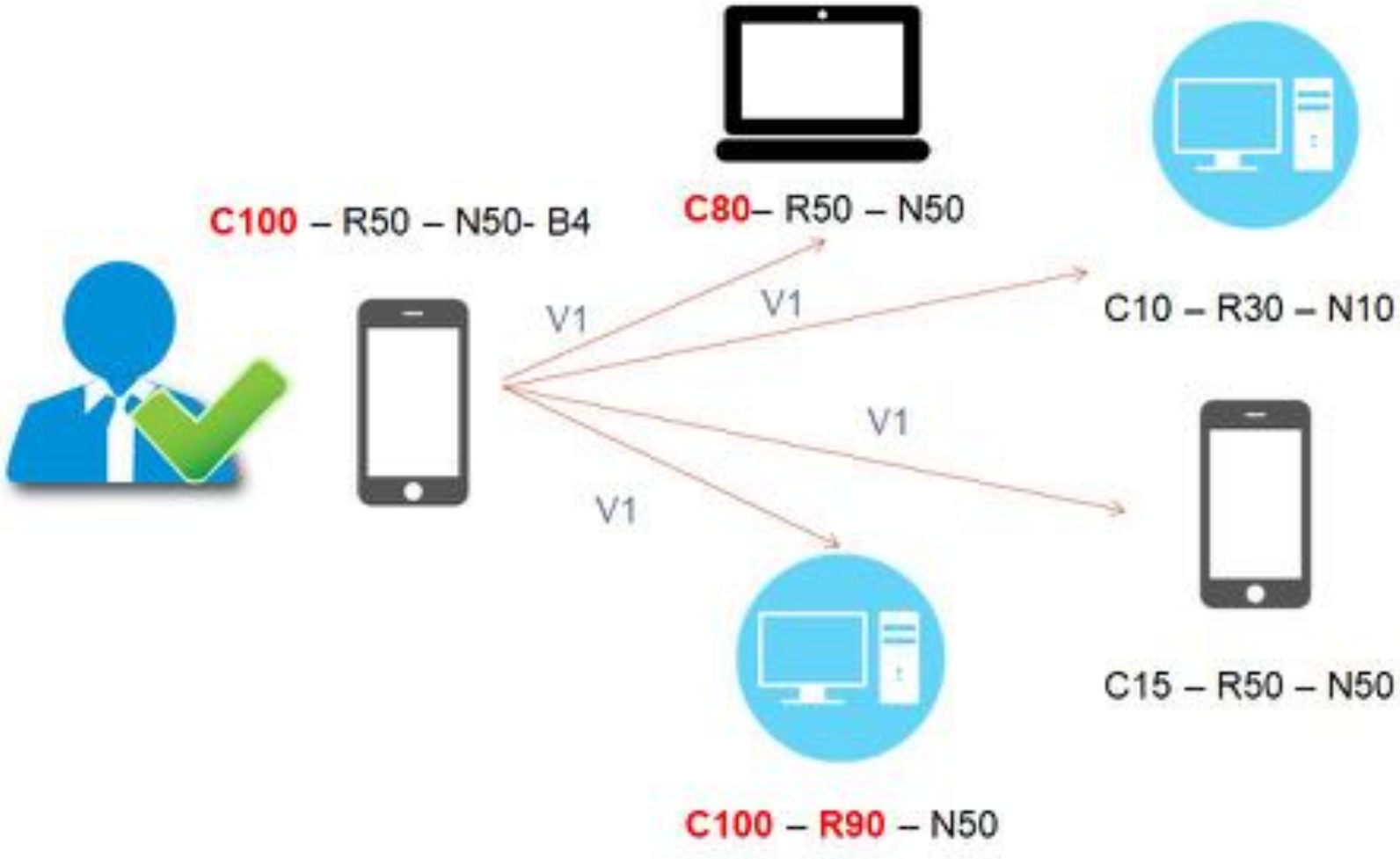
C100 – R50 – N50- B4

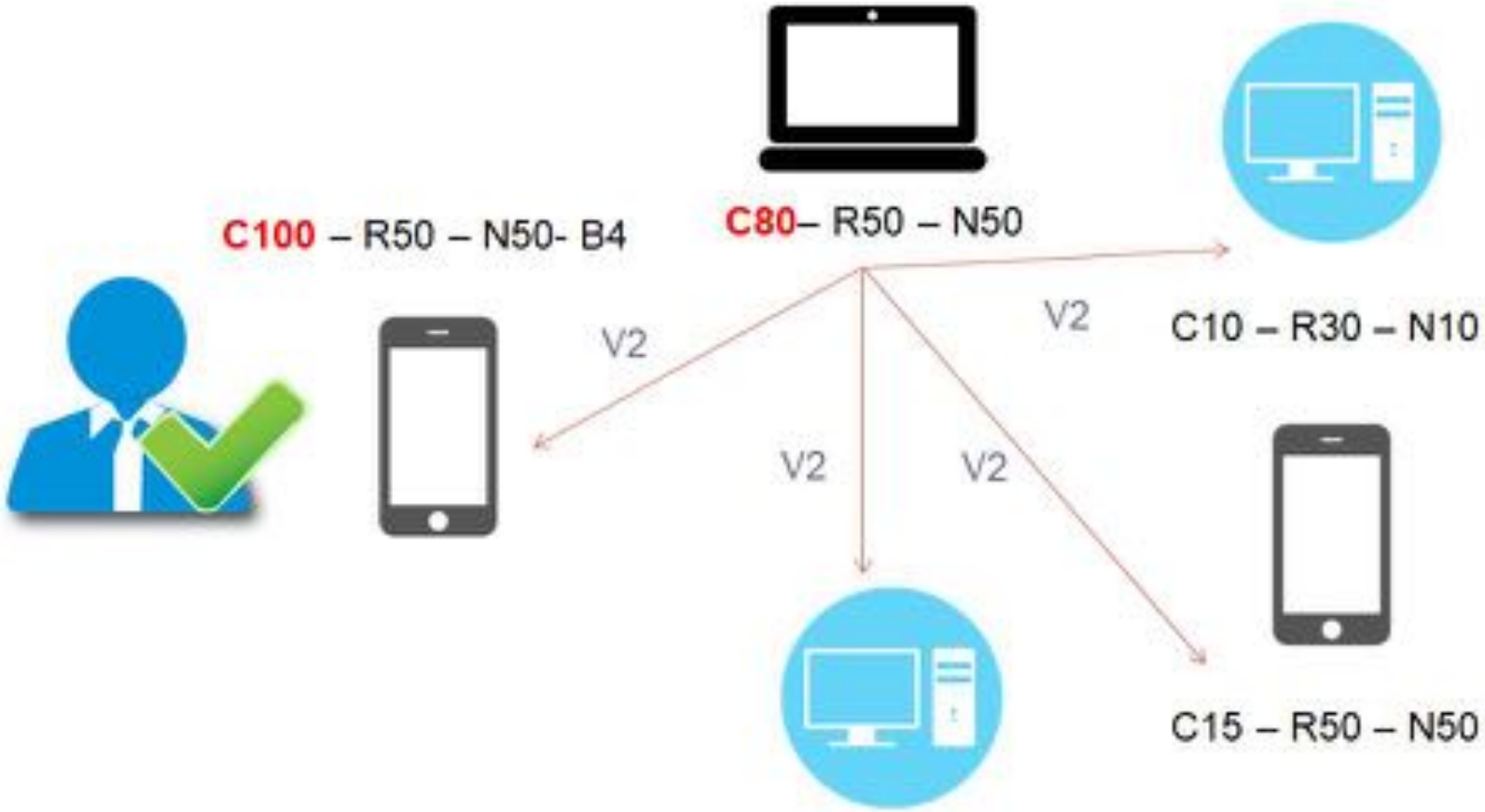
C80– R50 – N50

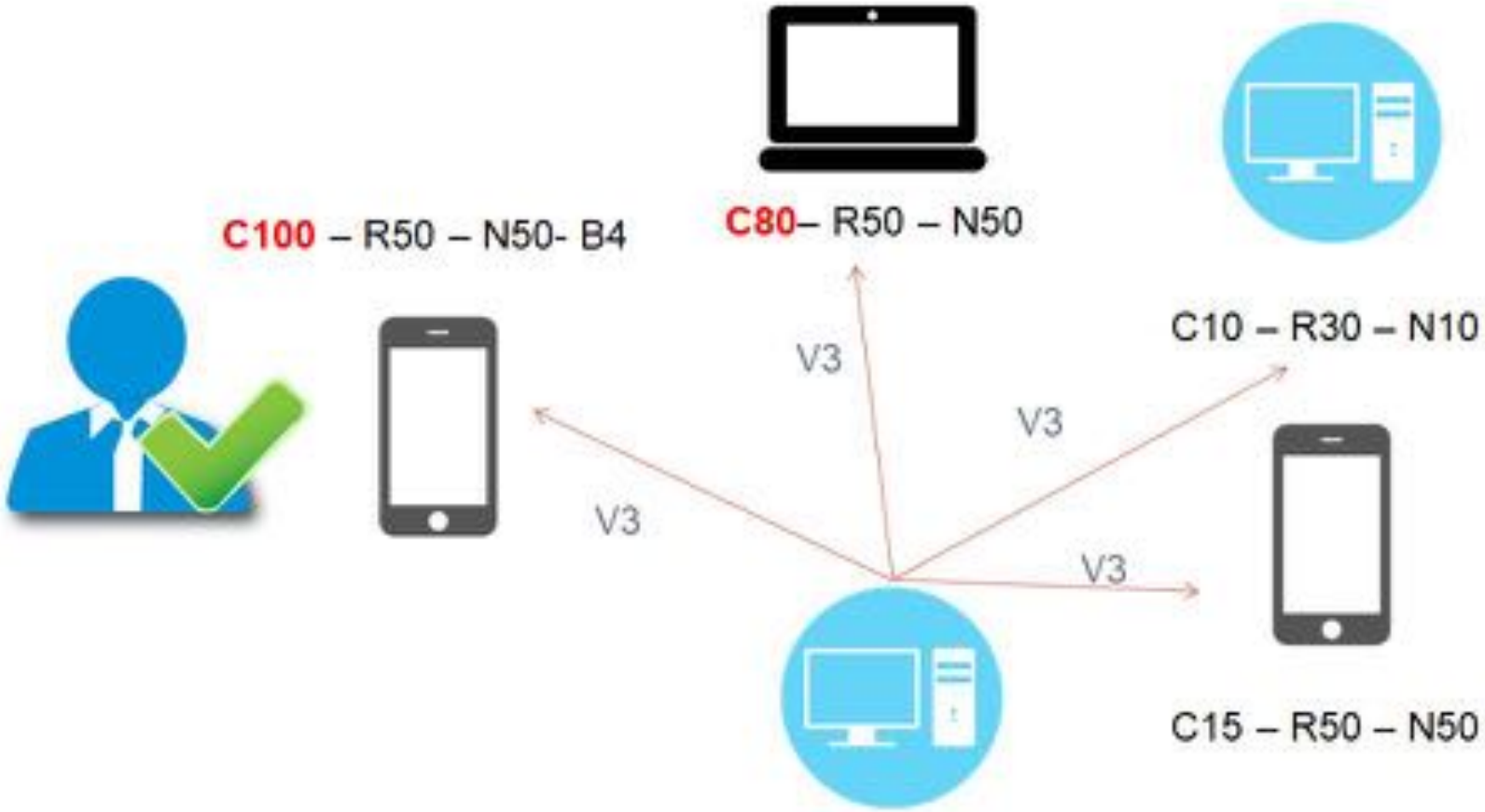
C10 – R30 – N10

C15 – R50 – N50

C100 – R90 – N50


C80– R50 – N50
C100 – R50 – N50- B4
V1
V1
V1
V1
C10 – R30 – N10
C15 – R50 – N50
C100 – R90 – N50


C80– R50 – N50
C100 – R50 – N50- B4
V2
V2
C10 – R30 – N10
V2
V2
C15 – R50 – N50
C100 – R90 – N50


C80– R50 – N50
C100 – R50 – N50- B4
C10 – R30 – N10
V3
V3
V3
V3
C15 – R50 – N50
C100 – R90 – N50

C100 – R50 – N50- B4

C80– R50 – N50

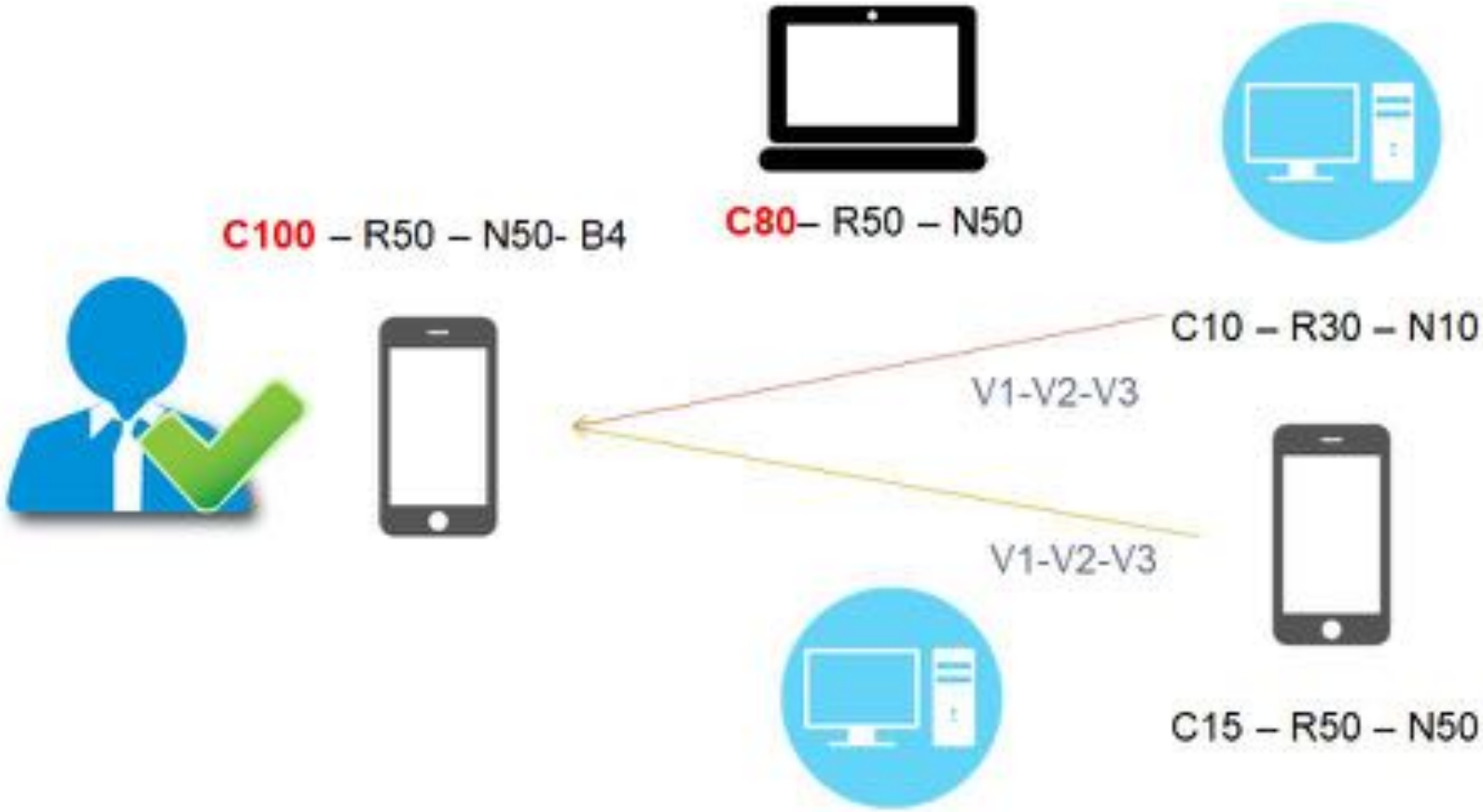
C10 – R30 – N10

V1-V2-V3

V1-V2-V3

C15 – R50 – N50

C100 – R90 – N50


C80– R50 – N50
C100 – R50 – N50- B4
C10 – R30 – N10
V1-V2-V3
V1-V2-V3
C15 – R50 – N50
C100 – R90 – N50

C100 – R50 – N50- B4

C80– R50 – N50

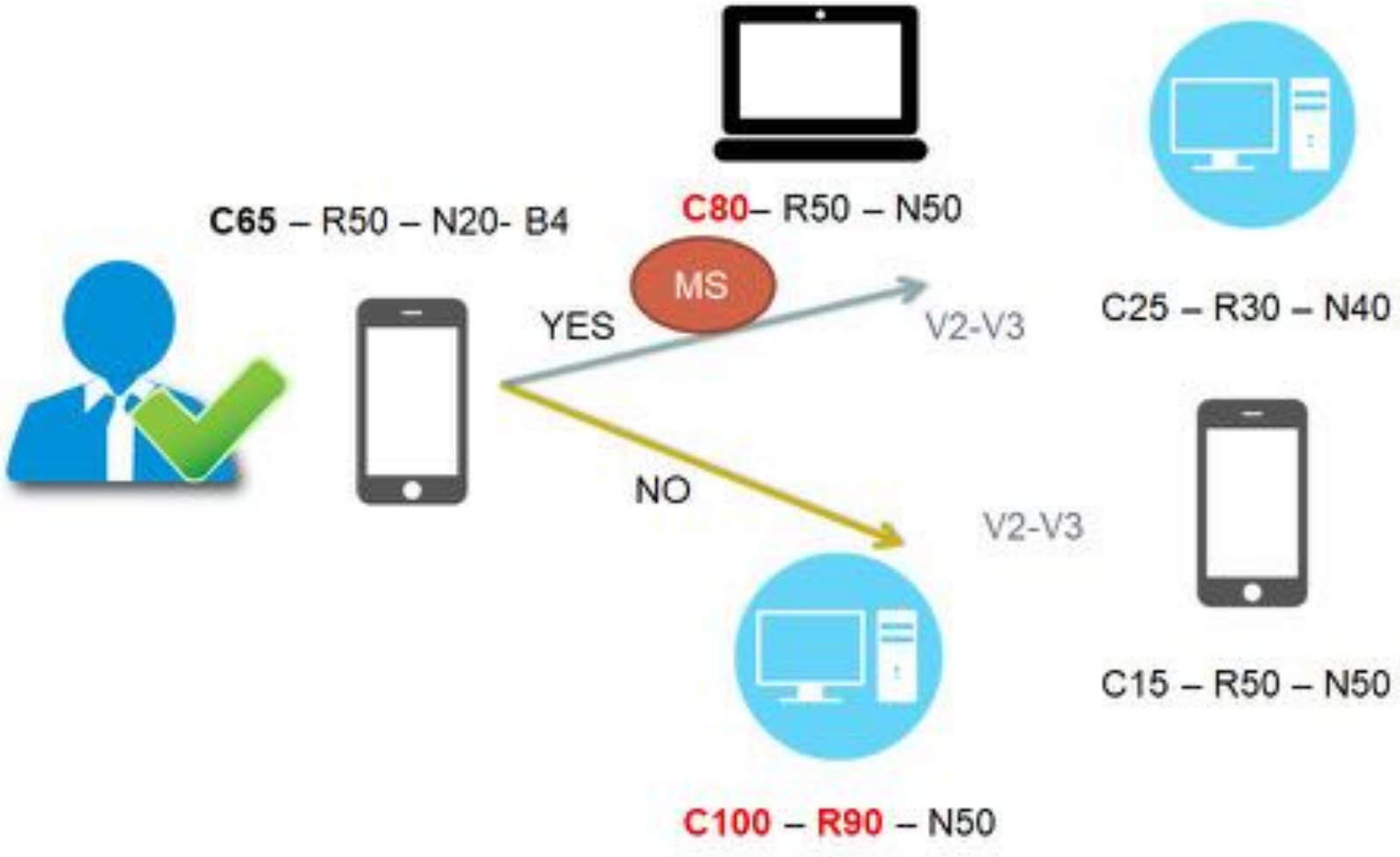
C10 – R30 – N10

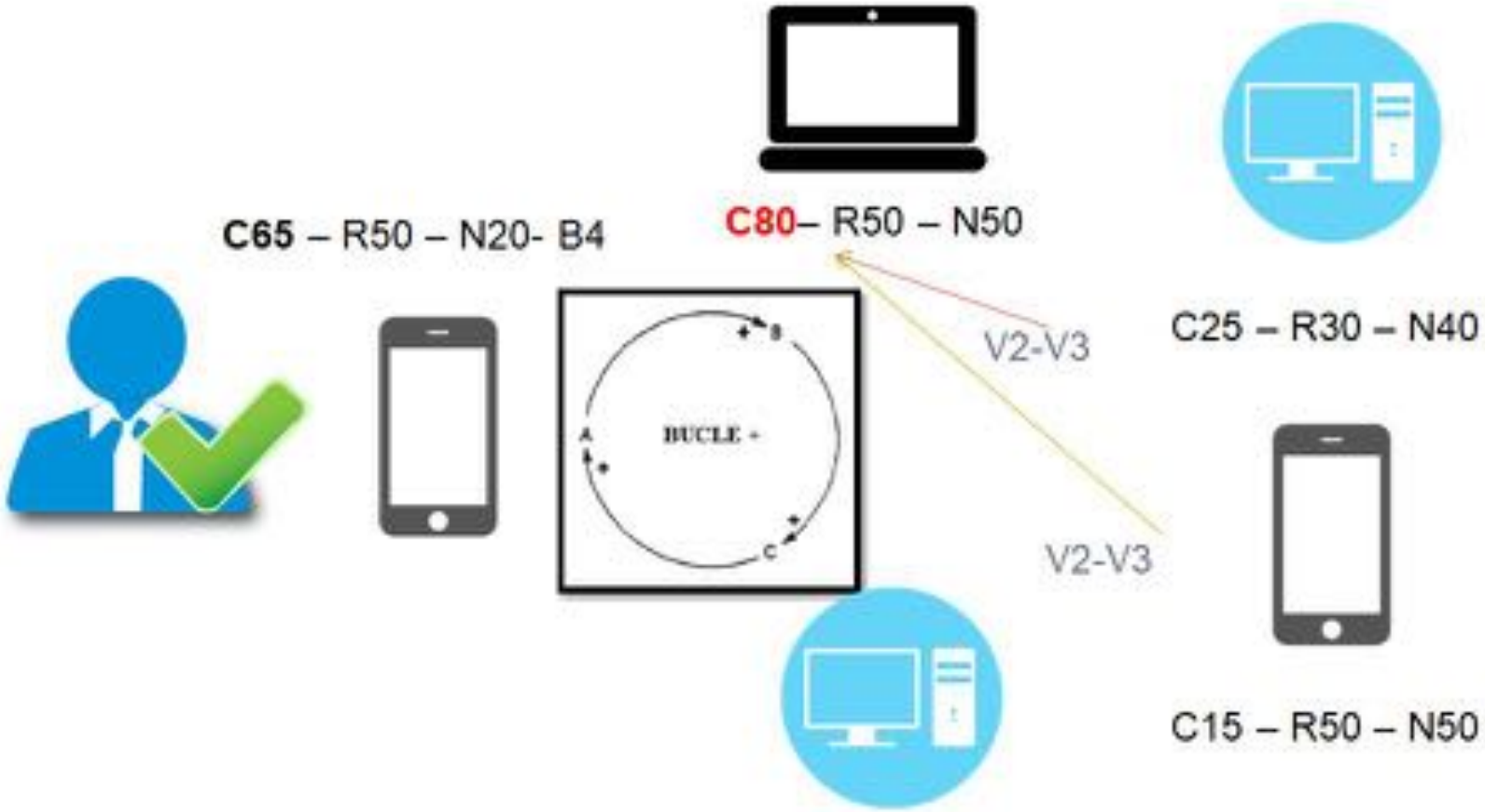
V1-V2-V3

V1-V2-V3

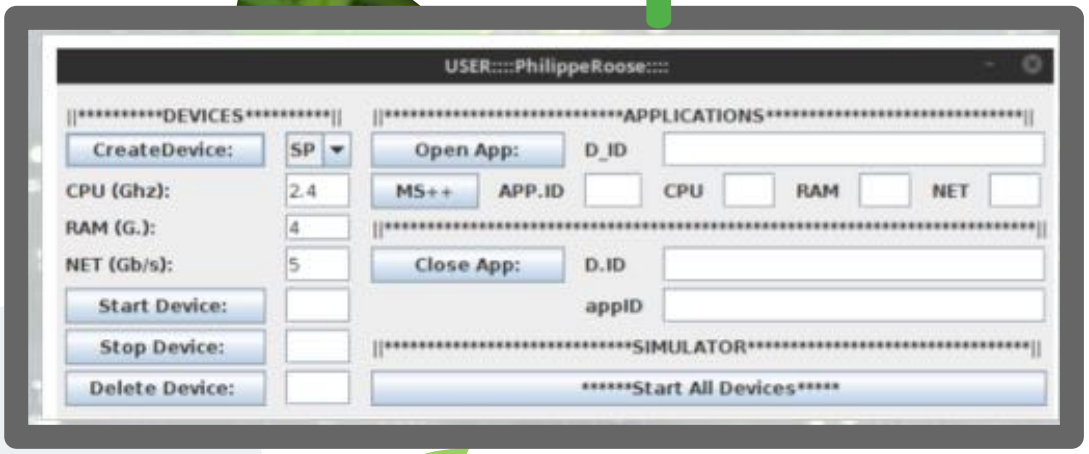
C15 – R50 – N50

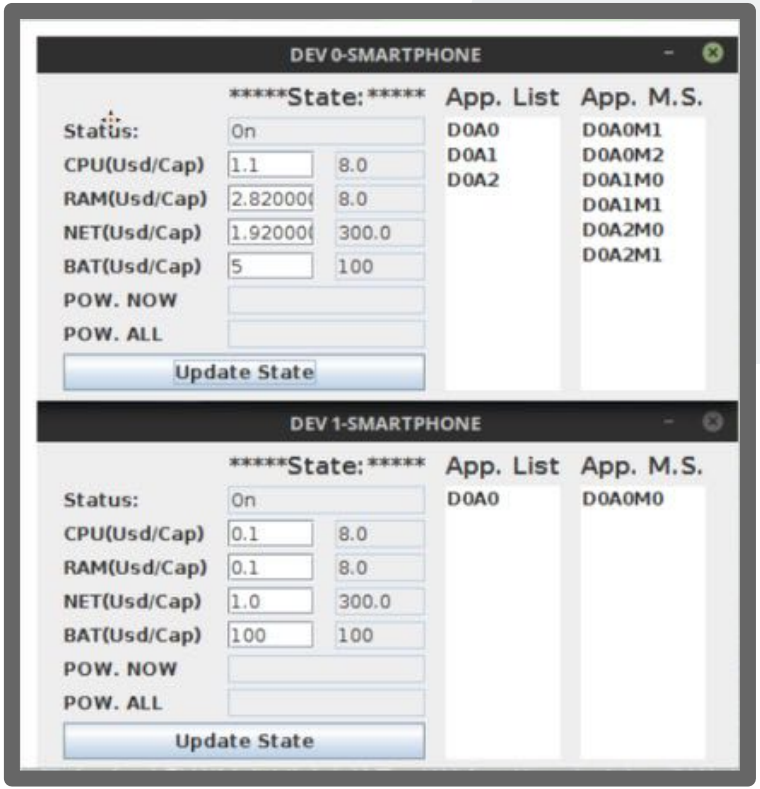
C100 – R90 – N50


C80– R50 – N50
C65 – R50 – N20- B4
MS
YES
V2-V3
C25 – R30 – N40
NO
V2-V3
C15 – R50 – N50
C100 – R90 – N50


C80– R50 – N50
C65 – R50 – N20- B4
BUCLE +
V2-V3
V2-V3
C25 – R30 – N40
C15 – R50 – N50
C100 – R90 – N50

Implementation...






```
{"appID":"D0A0","msID":"D0A0M0","CPU":"1","RAM":"1","NET":"1"}
{"appID":"D0A0","msID":"D0A0M1","CPU":"2","RAM":"1","NET":"2"}
{"appID":"D0A0","msID":"D0A0M2","CPU":"1","RAM":"3","NET":"3"}
{"appID":"D0A1","msID":"D0A1M0","CPU":"1","RAM":"1","NET":"4"}
{"appID":"D0A1","msID":"D0A1M1","CPU":"3","RAM":"2","NET":"5"}
{"appID":"D0A2","msID":"D0A2M0","CPU":"1","RAM":"2","NET":"4"}
{"appID":"D0A2","msID":"D0A2M1","CPU":"3","RAM":"1","NET":"5"}
{"appID":"D0A2","msID":"D0A2M2","CPU":"2","RAM":"1","NET":"5"}
{"appID":"D0A2","msID":"D0A2M3","CPU":"1","RAM":"1","NET":"5"}
{"appID":"D0A2","msID":"D0A2M4","CPU":"2","RAM":"2","NET":"40"}
```

```
{"msID":"D0A2M0","CONNECTION":"D0A2M3","TRATE":"200"}
{"msID":"D0A2M1","CONNECTION":"D0A2M3","TRATE":"500"}
{"msID":"D0A2M2","CONNECTION":"D0A2M3","TRATE":"700"}
{"msID":"D0A2M3","CONNECTION":"D0A2M4","TRATE":"800"}
```

Results...

Number of Devices	MS per Dvice	Avg. Applications execution time before the Algorithm	Avg. Applications execution time after the Algorithm.
2	150	2.335	2.4
3	100	2,4	2,43
4	75	2.45	2.99
5	60	2,43	3.71
6	50	2.44	3.76

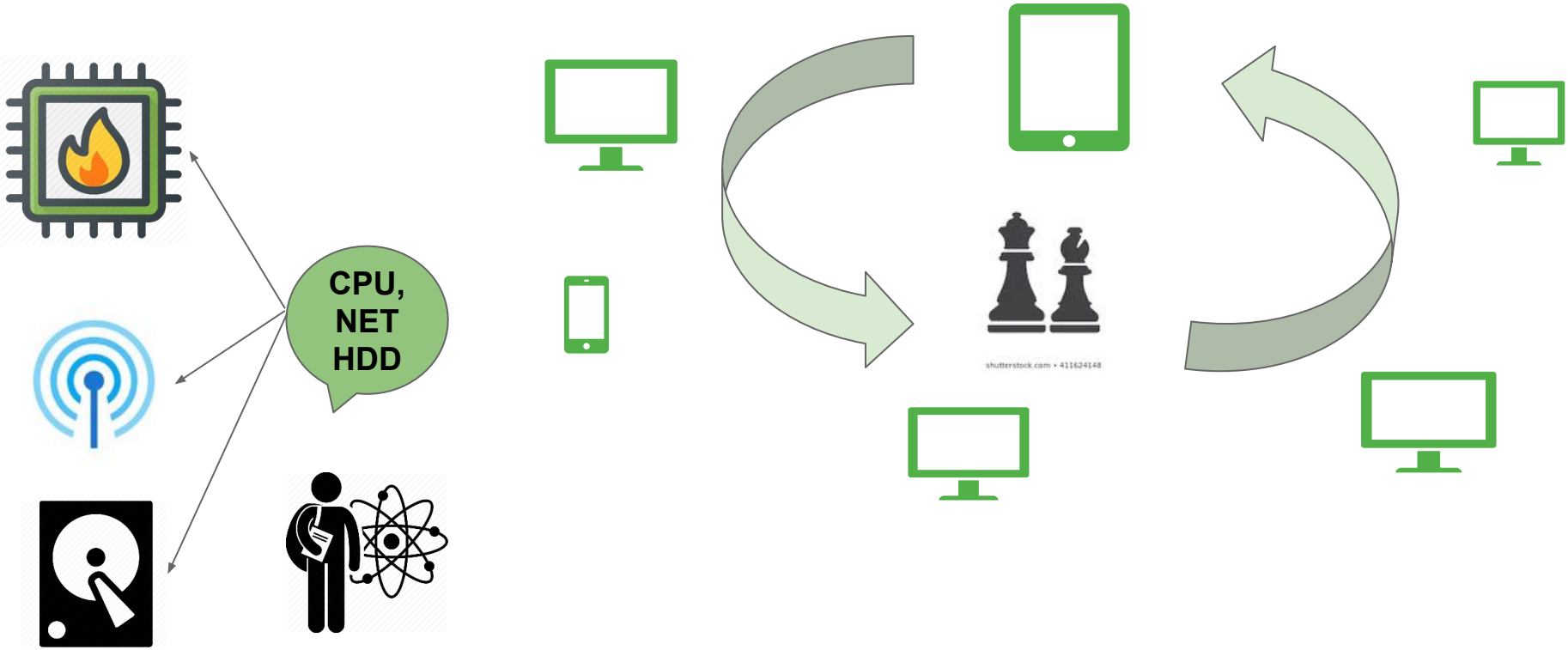
Kaligreen: Pros and Cons

PROS	CONS
- Scalable - Increases the execution time of applications on devices with battery and improve battery usage. - Does not overload the network - Decentralized and autonomous.	- Does not consider device disconnection situations. - Does not prevent infinite cyclical reconfigurations. - Does not study the opportunities of hardware components. - Does not consider the user's actual needs - Does not save energy...

5. Kaligreen V2

Kaligreen is now capable to decide

- How to deploy an application (ie. where to deploy microservices - which host devices)
- Considers **hardware component offers (will be explained in the next slide).**
- Kaligreen **can decide to move and duplicate** microservices to save energy.



Boosting, PCPG, FVFS!

Microservice Features		CPU features		
Persistent Microservice	**High CPU Consumption**	**Boosting**	**PCPG**	**DVFS**
NO	YES	Candidate	Candidate	Candidate
NO	NO	--	--	--
YES	YES	Candidate	Candidate	Candidate
YES	NO	--	Candidate	Candidate

THE CPU

THE NETWORK

I can save energy too: Reduce power; but it is not important at middleware level

Microservice Features					Network operations		
Persistent MS	Heavy MS	Use a lot of Bandwidth	A lot of dependencies with others MS and Data in device	large MS data	Move MS	Duplicate MS	Move MS Data
YES	YES	YES	YES	YES	Candidate	Candidate	--
YES	YES	YES	YES	NO	Candidate	Candidate	Candidate
YES	YES	YES	NO	YES	Candidate	Candidate	--
YES	YES	YES	NO	NO	Candidate	Candidate	Candidate
YES	YES	NO	YES	YES	--	Candidate	--
YES	YES	NO	YES	NO	--	Candidate	Candidate
YES	YES	NO	NO	YES	Candidate	Candidate	--
YES	YES	NO	NO	NO	Candidate	Candidate	Candidate
YES	NO	YES	YES	YES	Candidate	Candidate	--
YES	NO	YES	YES	NO	Candidate	Candidate	Candidate
YES	NO	YES	NO	YES	Candidate	Candidate	--

THE HARD DISK

I can save energy if I'm off!

Condition	Action
if Hdd requirements of Application M.S. == total load of hard disk now	Candidate to move (Hdd will be able to turn off itself)

Algorithm 1 Algorithm 1: Selecting candidate microservices

1: $L\_M \leftarrow List\_of\_all\_Microservices$
2: **while** *true* **do**
3: $L\_CPU \leftarrow filter\_by\_Table1(L\_M)$
4: $L\_DISK \leftarrow filter\_by\_Table3(L\_M)$
5: $L\_NETWORK \leftarrow filter\_by\_Table2(L\_M)$
6: $SLEEP(T)$
7: **end while**

> *Then, Kaligreen can order the lists according to CPU, network, disk and overall microservice consumption*

$$MS\_Cons = T*(F(N+extern)+F(D)+F(C)) \quad (1)$$

In this moment, we are working on a scheduling algorithm based on P2P, graph theory and statistical approaches

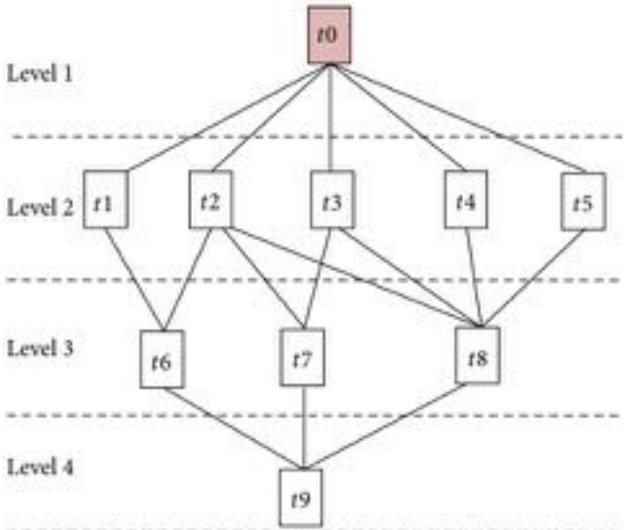


Figure donne by Piyush Chauhan and Nitin: Decentralized Scheduling Algorithm for DAG Based Tasks on P2P Grid

Thanks!

Time for questions!...