Gossip-based Resource Allocation for Green Computing in Large Clouds

Rerngvit Yanggratoke Fetahi Wuhib Rolf Stadler

ACCESS Linnaeus Center
KTH Royal Institute of Technology, Sweden

CNSM2011

October 27, 2011
Paris, France

Motivation

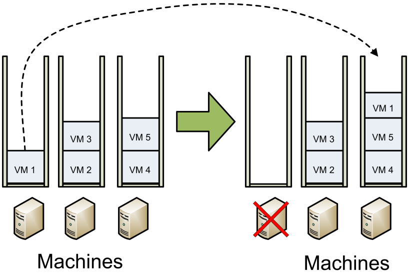
Microsoft's 303'000 sq.ft. cloud datacenter in Dublin, IE.

"Data center around the world consume 201.8 terawatt hours (TWh) and energy expenditures reached $23.3 billion in 2010"

–Pike Research, 2010.

Approach: Server Consolidation

Minimizing number of active servers. Idle servers can be standby.



Why?

- Servers in datacenters are often underutilized because they are overprovisioned for spike traffic.
- An idle server consumes at least 60% of its power consumption under full load (VMWare DPM, 2010).

Enabling technology: Live migration, volumn migration, various levels of standby mode.

Related work

- Products : VMWare DPM, Ubuntu Entreprise Cloud Power Management.
- Research :
  - Static consolidation solution: (Verma et al., 2009), (Cardosa, et al., 2009), (Speitkamp and Bichler, 2010), (Subramanian et al., 2010)
  - Dynamic consolidation solution: (Jung et al., 2010), (Gmach, et al., 2008)
  - Combined energy efficiency solution: (Petrucci et al., 2010), (Tolia et al., 2009)

- All of them based on some centralized solutions that have well-known problem regarding scalability.

Design goals and design principles

Design goals for Resource Management System

- **Server consolidation** in case of underload.
- **Fair** resource allocation in case of overload.
- **Dynamic adaptation** to changes in load patterns.
- **Scalable operation** ($> 100,000$ machines).

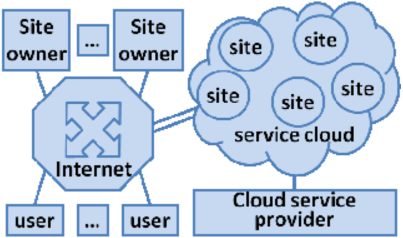
Design principles

- A distributed **middleware architecture**.
- **Distributed protocols** - gossip-based algorithms.
  - Generic protocol for resource management(**GRMP**).
  - Instantiation for solving the goals above(**GRMP-Q**).

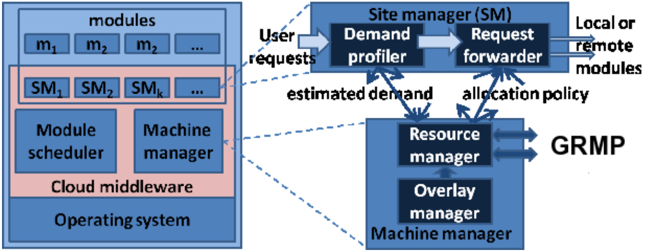
The use case – PaaS

- The cloud service provider operates the physical infrastructure.
- The cloud hosts **sites** belonging to its clients.
- Users access sites through the Internet.
- A site is composed of **modules**.
- Our focus: allocating **CPU** and **memory** resources to sites.

The stakeholders.

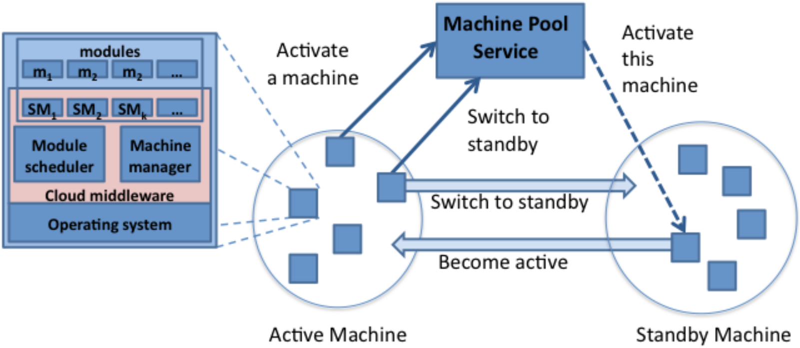


Middleware architecture



The middleware runs on all machines in the cloud.

F. Wuhib, R. Stadler, M. Spreitzer: "Gossip-based Resource Management for Cloud Environments," 6th International Conference on Network and Service Management, Niagra Falls, Canada, October 2010.



The machine pool service.

Examples: Standby - ACPI G2(Soft-off).
Activate - wake-on-LAN (WoL) packet.

Modeling resource allocation

Demand and capacity

M, N: set of modules and machines (servers) respectively.
$\omega_m(t)$, γ_m: CPU and memory demands of module $m \in M$.
Ω, Γ: CPU and memory capacity of a machine in the cloud.

Resource allocation

$\omega_{n,m}(t) = \alpha_{n,m}(t)\omega_m(t)$: demand of module m on machine n.
$A(t) = (\alpha_{n,m}(t))_{n,m}$ a **configuration matrix**.

Machine n allocates $\hat{\omega}_{n,m}(t)$ CPU and γ_m memory to module m.
according to **local resource allocation policy** $\hat{\Omega}$:

$$\hat{\omega}_{n,m}(t) = \Omega \, \omega_{n,m}(t) / \sum_i \omega_{n,i}$$

.

Modeling utility and power consumption

Utility

$u_{n,m}(t) = \frac{\hat{\omega}_{n,m}(t)}{\omega_{n,m}(t)}$: utility of module m on machine n.
$u(s,t) = \min_{n,m \in M_s} u_{n,m}(t)$: site utility.
$U^c(t) = \min_{s | u(s,t) \leq 1} u(s,t)$: cloud utility.

Power consumption

Assuming homogenous machines,

$$P_n(t) = \begin{cases} 0 & \text{if } row_n(A)(t)\mathbf{1} = 0 \\ 1 & \text{otherwise} \end{cases}$$

The resource allocation problem

Resource allocation as a utility maximization problem

$$\begin{aligned}
\text{maximize} \quad & U^c(t+1) \\
\text{minimize} \quad & P^c(t+1) \\
\text{minimize} \quad & c^*(A(t), A(t+1)) \\
\text{subject to} \quad & A(t+1) \geq 0, \;\; \mathbf{1}^T A(t+1) = \mathbf{1}^T \\
& \hat{\Omega}(A(t+1), \boldsymbol{\omega}(t+1))\mathbf{1} \preceq \mathbf{\Omega} \\
& \mathbf{sign}(A(t+1))\boldsymbol{\gamma} \preceq \mathbf{\Gamma}.
\end{aligned}$$

Cost of reconfiguration

c^* can be the number of module instances that are started to reconfigure the system.

Protocol GRMP: pseudocode for machine n

initialization

read $\boldsymbol{\omega}, \gamma, \Omega, \Gamma, row_n(A)$;
$initInstance();$
start passive and active threads;

active thread

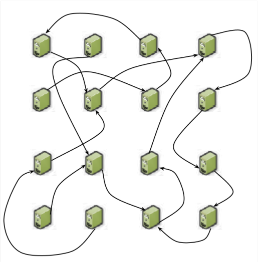
for $r = 1$ to r_{max} **do**
  $n' = choosePeer();$
  $send(n', row_n(A));$
  $row_{n'}(A) = receive(n');$
  $updatePlacement(n', row_{n'}(A));$
  sleep until end of round;
write $row_n(A)$;

passive thread

while $true$ **do**
  $row_{n'}(A) = receive(n');$
  $send(n', row_n(A));$
  $updatePlacement(n', row_{n'}(A));$

Three abstract methods:

- $initInstance();$
- $choosePeer();$
- $updatePlacement(n', row_{n'}(A));$



Protocol GRMP-Q: pseudocode for machine n

initInstance()

```
read N_n;
```

choosePeer()

```
if rand(0..1) < p then
    n' = unifrand(N_n);
else
    n' = unifrand(N − N_n);
```

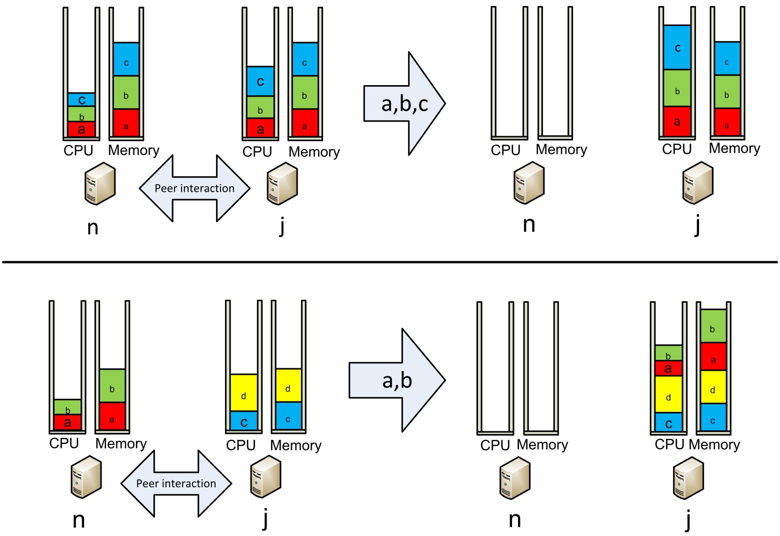
- Prefer a gossiping peer with common modules.
  $N_n = \{j \in N$, where j has common modules with $n\}$.

updatePlacement($j, row_j(A)$)

```
if (ω_n + ω_j ≥ 2Ω) then
    equalize(j, row_j(A));
else
    if j ∈ N_n then
        packShared(j);
    packNonShared(j);
```

- Equalize if aggregate load $\geq$ aggregate capacity.
- Pick destination machine to pack:
  - machine with higher load if both are underloaded.
  - machine in underloaded machine if one is overloaded.

GRMP-Q - method updatePlacement$(j, row_j(A))$



Properties of GRMP-Q

- Case CPU overload, sufficient memory:

  the protocol achieves a fair allocation(min-max fairness).

- Case CPU underload, sufficient memory:

  The protocol converges into a configuration where minimum number of machines is active.

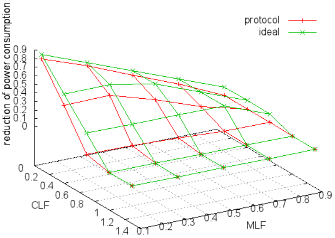
- Other cases:

  As long as there is an available standby machine, the protocol guarantees that the demand of all sites is satisfied.

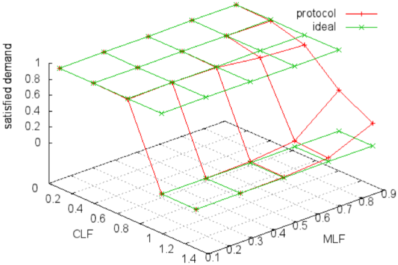
Simulation: demand, capacity and evaluation metrics

- Demand ω changes at discrete points in time at which GRMP-Q recomputes A.
- **Demand**: CPU demand of sites is Zipf distributed. Memory demand of modules is selected from {128MB, 256MB, 512MB, 1GB, 2GB}.
- **Capacity**: CPU and memory capacities are fixed at 34.513 GHz and 36.409GB respectively.
- **Evaluation scenarios**
  - different CPU and memory load factors (CLF, MLF). CLF={0.1, 0.4, 0.7, 1.0, 1.3} and MLF={0.1, 0.3, 0.5, 0.7, 0.9}.
  - different system size.
- **Evaluation metrics**: reduction of power consumption, fairness, satisfied demand, cost of reconfiguration.

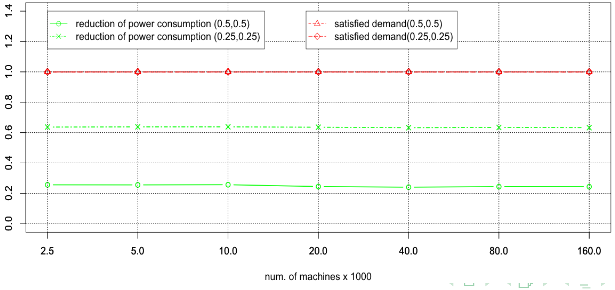
Measurement Results



(a) Fraction of machines that can be put to standby.



(b) Fraction of sites with satisfied demand.



(c) Scalability with respect to the number of machines and sites.

Conclusion

- We introduce and formalize the resource allocation problem with green objective.
- We develop **GRMP, a generic gossip-based protocol for resource management** that can be instantiated with different objectives.
- We develop **an instance of GRMP which we call GRMP-Q**, and which provides a heuristic solution to the server consolidation problem. This is the second instance of GRMP.
- We perform a simulation study of the performance of GRMP-Q, which indicates that the protocol qualitatively behaves as expected based on its design.
- Implementation in Openstack is in progress.

Future work

Works relating to resource allocation protocol GRMP-Q:

- Extend the model to capture other resources including storage and network.
- Improve its convergence property for large CLF values.
- Develop a support for heterogeneous machines.
- Customize so that it would have robustness regarding failures.

Regarding the middleware architecture:

- Design a mechanism for deploying new sites.
- Scalable design of the machine pool manager.
- Extend design to span multiple clusters / data centers.

Ongoing implementation of the protocol.