International Journal of Electrical and Computer Engineering (IJECE)
Vol.3, No.1, February 2013, pp. 21~29
ISSN: 2088-8708

Theoretical Analysis on Scale-down Aware Service Allocation in Cloud Storage Systems

Angli Liu
Department of Electronic Engineering, Tsinghua University

Article Info

Article history:

Received Nov 13, 2012
Revised Jan 12, 2013
Accepted Jan 24, 2013

Keyword:

Cloud computing
Service scheduling
Energy efficiency
Probability partitioning
Workload Model

ABSTRACT

Servcie allocation algorithms have been drawing popularity in cloud computing research community. There has been lots of research on improvingservice allocation schemes for high utilization, latency reduction and VM migration enfficient, but few work focus on energy consumption affected by instance placement in data centers.

In this paper we propose an algorithm in which to maximize the number of freed-up machines in data centers, machines that host purely scale-down instances, which are reuiqred to be shut down for energy saving at certain points of time. We intuitively employ a probability partitioning mechanism to schedule services such that the goal of the maximization can be achieved. Furthermore we perform a set of experiments to test the partitioning rules, which show that the proposed algorithms can dynamically increase the number of freed-up machines substantially.

Corresponding Author:

Angli Liu,
Departement of Electronic Engineering,
Tsinghua University,
Beijing, 100084, China.
Email: xbeiba@gmail.com

1. INTRODUCTION

We consider such a problem where services are continuously inserted into a one-cluster data center. In each service, the instance# in each UD$N_{InstancePerUD}$ is subject to a certain distributionF($N_{InstancePerUD}$),say U(1, 10), the UD# in each service N_{UD} is subject to another distribution F(N_{UD}), say U(1, 5), and the UD is assigned to each instance in a round-robin fashion. We would like to figure out to which node we allocate each instance such that the expected number of freed-up nodes, nodes that host purely scale-down instances, can be maximized.

For the allocation coordinator (AC) in the data center, the known parameters are the index, size, UD, FD of each instance and the probability distribution of the scale-down threshold instance index in each service, meaning that instances with indices larger than this probabilistically set threshold will scale down and instances with indices lower will not.

On the other hand, what the AC does not exactly is whether an instance is a scale-down one or not; it will be shown that the AC, in turn, can know the scale-down probability of an instance. Therefore, we need to use the known parameters and distributions to design some rules for the AC to allocate instances in order to maximize the freed-up node count.

2. PROBLEM FORMULATION

Intuitively, should instances with high scale-down probability be grouped and instances with low scale-down probability are gathered in other groups of nodes, the expectation of freed-up node count will be

larger than just randomly allocate instances. So the problem is divided into two parts: how many groups, or segments, should we use, and how to partition the probability line into segments. Basically we can tackle these two sub-problems independently. First we can use a fixed number of segments for partitioning in order to find out the rules for grouping instances, or partitioning the probability line, and then we can study the impact of the number of segments on the expected number of freed-up nodes.

2.1. Probability Partitioning

A partitioning rule can be illustrated as in Fig. 1, where the x-axis stands for machine count in a segment, and the y-axis stands for the probability line.

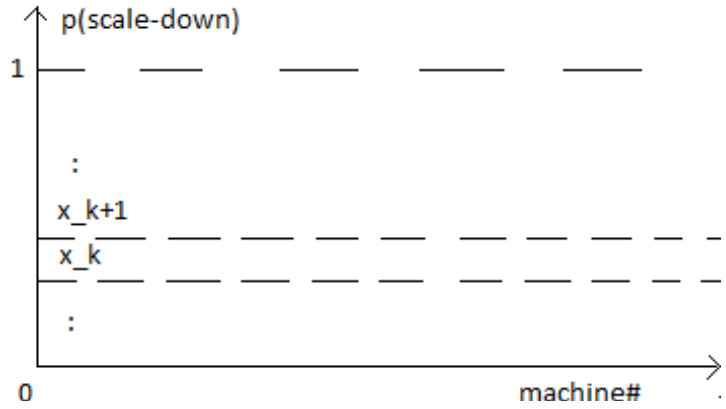

Figure 1. Partitioning the probability line

Having this been illustrated, we would further like to know what the proportion of the instances is that will be assigned to a certain segment, formulated by (x_k, x_{k+1}), so that the number of nodes, especially freed-up nodes in each segments can be further determined using this instance number.

2.2. Scale-down probability distribution in terms of indices

To work out this instance number, we can first bridge the index of an instance in its host service and the instance scale-down probability. Provided that the scale-down threshold index distribution curve, noted as $T_j(x)$, is known, we can do the integral on this curve from 0 to the index of the instance we consider, say i, and the result of the integral will be the scale-down probability of the instance with this index in this particular service, as shown in Fig. 2, where j is for labeling the service size.

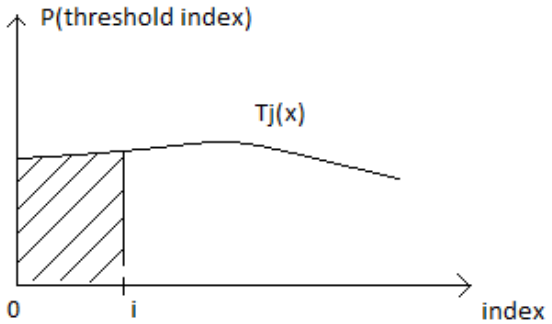

Figure 2.The calculation of the scale-down probability of an instance in a service scale

If we want to know the general scale-down threshold probability in terms of an index, we need to know the distribution of instance number (maximum index) in a service and weight it on the scale-down probability of an instance with this index in a service. So we would like to exactly know the relationship between the probability of the occurrence of an index and an index, i.e., the probability density function of

instance index, so we can do the weighting. In fact, we already know the UD distribution and instance number distribution in a UD. And the instance index, which is the product of the UD count and instance count per UD, $N_{InstancePerUD} \times N_{UD}$, can be easily worked out using the two independent distributions. For example, if these two are subject to uniform distribution, then this product subject to

$$P(N_{Instance} < N) = P\left(N_{UD} < \frac{N}{N_{InstancePerUD}}\right) = \sum_{j=1}^{perUD\#} P(N_{UD} < \frac{N}{j} | N_{InstancePerUD} = j) P(N_{perUD} = j)$$
$$= \sum_{j=N/UD\#}^{InstancePerUD\#} \frac{N}{j} \frac{1}{UD\#}$$

as shown in Fig. 3. We note this curve as $s(k)$.

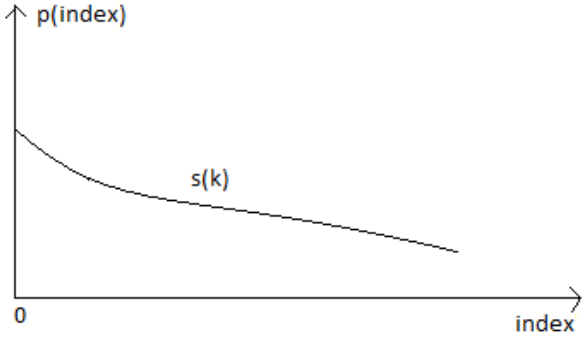

Figure 3. Index distribution

To get the general scale-down threshold probability distribution Q(x) in terms of indices, we just need to weight each $s(j)$ on $T_j(x)$ and sum them up, as follows

$$Q(x) = \sum_j s(j) T_j(x)$$

Then, as in Fig. 2, we do the integral for $Q(x)$ and get the general scale-down probability distribution $P(x)$.

$$P(x) = \int_0^x Q(y) dy$$

With this and the probability partitioning as described in A, we can exactly map a certain probability segment to an instance index range, as shown in Fig. 4.

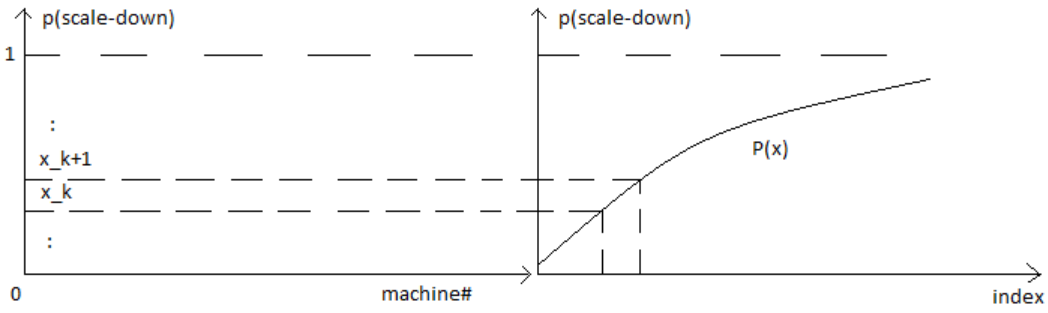

Figure 4. Mapping a scale-down probability segment to an instance index range

2.3. The proportion of instances assigned to a certain scale-down probability segment

With the above mapping and the index distribution shown in Fig. 3, we can actually work out the proportion of incoming instances that will be assigned to a certain probability segment, as shown in Fig. 5, where we calculate the area of the shadow area to get this proportion.

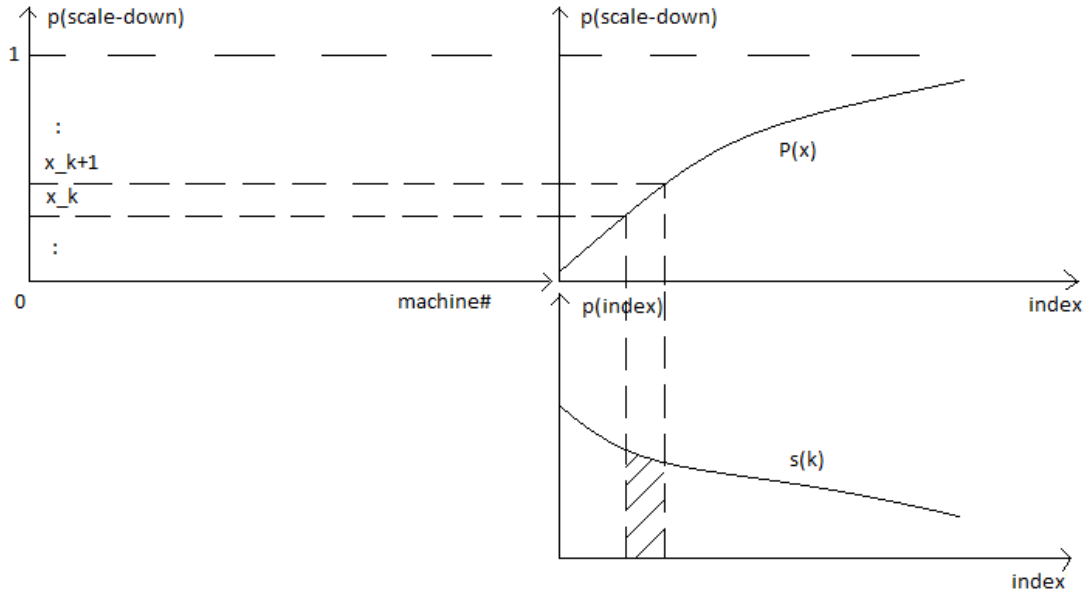

Figure 5. Instance proportion mapping

2.4. UD count in each segment

It is not enough if we just stop here, because we have not worked out the freed-up node count in each segment, which is the final determinant to whether a partitioning rule is good or bad. However, with the instance proportion, we are actually able to work the freed-up node count out, using the relationship between instance index and the UD count, which largely determines the node count in a probability segment.

What we are exactly doing is working out the UD count when the instance index is provided, or a range of instance indices is given. Using Bayes' Formula, we can work this out, as presented below

$$P(UD\# = n|index = k) = \frac{P(UD\# = n)P(index = k|UD\# = n)}{P(index = k)}$$

In the above equation, $P(index = k)$ is given by $s(k)$, which we already have in B, $P(UD\# = n)$ is known initially, and $P(index = k|UD\# = n)$ can be derived by adding several independent UD# distributions together. This could be easy if UD# in a service is subject to uniform distribution. Now we draw out $U(x)$, i.e., $P(UD\# = n|index = k)$, the relationship between index and UD#, as in Fig. 6.

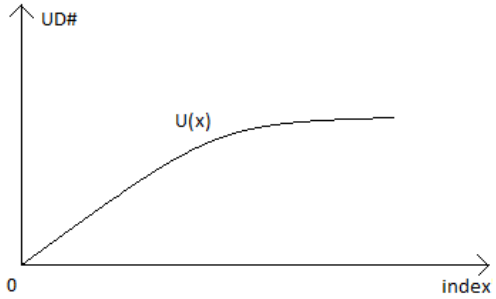

Figure 6. The relationship between UD# and instance index

Conclusively, using a probability segment, we are able to know the proportion of incoming instances that will be assigned to this segment. The entire mapping is shown in Fig. 7.

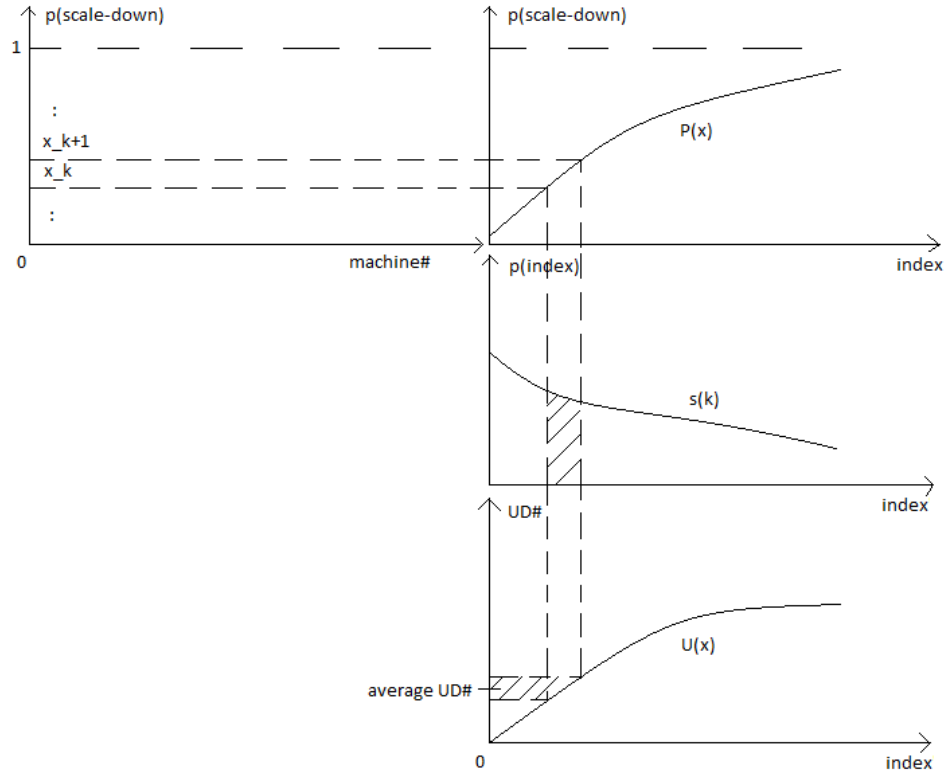

Figure 7. Mapping the probability segments to average UD#

2.5. *Freed-up node count in each segment*

Because the final goal is to formulate the freed-up node count in each segment, and we are not there, we need to find out these two things: the node count in each segment and freed-up probability of a node in each segment.

The first one is able to be given by the average UD# in a segment. Since the node count is only determined by UD# and FD#, and FD is independent from UD, the node count is proportional to the average UD# with a coefficient β related to the FD constraint and the workload ingredient, as follows

$$\overline{N_{node}}(x_k, x_{k+1}) = \beta(FD, workload) \frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} U(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}}$$

As for the second one, the freed-up probability of a node, we need to figure out how possible on average does an instance in a segment scales down. Luckily, we already have Fig. 4, where the scale-down probability distribution is right mapped to scale-down probability partitioning. As a result, the probability of an instance assigned to segment (x_k, x_{k+1}) to scale down on average is given by

$$\overline{P(scale-down)} = \frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} P(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}}$$

Taking into account the average size of an instance $\overline{size}$, depending on the workload ingredient, we can get the "expected" freed-up probability of a node in segment (x_k, x_{k+1})

$$P(freed-up) = \overline{P(scale-down)}^{\frac{\overline{filled\ core}}{size}} = \left(\frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} P(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}}\right)^{\frac{\overline{filled\ core}}{size}}$$

$$= \left(\frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} P(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}}\right)^{\alpha(throttle,\ workload)}$$

whereα is only related to the workload ingredient the utilization throttle.
In summary, in segment (x_k, x_{k+1}), the average freed-up node count is given by

$$\overline{N_{freed-up}}(x_k, x_{k+1}) = \beta \cdot \frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} U(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}} \cdot \left(\frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} P(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}}\right)^{\alpha}$$

Where α and β is related to the workload ingredient, the FD constraint and the utilization throttle. Finally, the expectation of freed-up node count in the entire data center is given by

$$N_{freed-up} = \sum_k \overline{N_{freed-up}}(x_k, x_{k+1})$$

$$= \beta \sum_k \frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} U(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}} \cdot \left(\frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} P(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}}\right)^{\alpha}$$

$$= \beta \sum_k \frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} U(x)dx \cdot \left(\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} P(x)dx\right)^{\alpha}}{(P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)})^{\alpha+1}}$$

Where both $U(x)$and$P(x)$ are dependent on $s(k)$, the instance index distribution.

3. PROBLEM SOLVING

Using$N_{freed-up}$, theoretically we can determine each x_k by letting

$$\frac{\partial N_{freed-up}}{\partial x_k} = 0$$
$$k = 1, 2, \ldots, M-1$$

And the boundary conditions are

$$\sum_k \overline{N_{node}}(x_k, x_{k+1}) = \beta \sum_k \frac{\int_{P^{-1}_{(x_k)}}^{P^{-1}_{(x_{k+1})}} U(x)dx}{P^{-1}_{(x_{k+1})} - P^{-1}_{(x_k)}} = C$$
$$x_0 = 0$$
$$x_M = 1$$
$$0 < x_1 < \cdots < x_k < \cdots < x_{M-1} < 1$$

Where C is the total number of nodes in the data center, and M is the number of segments on the scale-down probability line.

After we work out these segment ends, we can use the following rules to allocate the incoming instances.

1) Based on the probability partitioning, assign a probability slot for this instance, and allocate it using PS-SUD algorithm within this segment. If the current machines cannot host this instance, request an additional empty machine.

2) If the utilization throttle is hit in this probability segment, request to allocate this instance onto another available (PS-SUD determined) machine that corresponds to the closet possible probability segment, in order to avoid big variance of scale-down probability of instances within the same segment.

4. EVALUATION

The above equations, while extracted by reasonable approximations, are of big difficulty to solve, and especially they are highly $U(x)$- and $P(x)$-oriented. Therefore, we propose these four potential solutions aiming to find out a better partitioning rule, corresponding to the four curves in Fig. 7.

1) Uniform Partitioning

Let $P(scale-down)$ be partitioned in a sense that in each $P(scale-down)$ segment, the differences of the two ends are equivalent, so as to achieve the "uniform partitioning".

2) Probability-equivalent Partitioning

Let $P(x)$ be partitioned in a sense that in each $P(x)$ segment, the areas between the two ends are equivalent, so as to achieve the "probability-equivalent partitioning".

3) Instance#-equivalent Partitioning

Let $s(k)$ be partitioned in a sense that in each $s(k)$ segment, the areas between the two ends are equivalent, so as to achieve the "instance#-equivalent partitioning".

4) Node#-equivalent Partitioning

Let $U(x)$ be partitioned in a sense that in each $U(x)$ segment, the areas between the two ends are equivalent, so as to achieve the "node#-equivalent partitioning".

Table 1. Segment# = 1

Algorithm	Freed-up node ratio/%	Scale-down instance ratio/%	Empty node ratio/%
Random	0	1.63	8.2

Table 2. Segment# = 2

Algorithm	Freed-up node ratio/%	Scale-down instance ratio/%	Empty node ratio/%
Uniform	0.6	17.20	3.8
Probability-equivalent	0.6	17.62	3.6
Instance#-equivalent	1.2	17.38	2.5
Node#-equivalent	3.6	17.61	2.1

Table 3. Segment# = 4

Algorithm	Freed-up node ratio/%	Scale-down instance ratio/%	Empty node ratio/%
Uniform	0.8	17.20	2.6
Probability-equivalent	0.9	17.62	3.5
Instance#-equivalent	1.8	17.39	2.1
Node#-equivalent	4.5	17.61	2.1

Table 4. Segment# = 8

Algorithm	Freed-up node ratio/%	Scale-down instance ratio/%	Empty node ratio/%
Uniform	1.3	17.20	1.1
Probability-equivalent	1.9	17.63	3.2
Instance#-equivalent	2.4	17.28	1.9
Node#-equivalent	9.3	17.59	0.5

We should note that in the above tables, empty node ratio less than 10% when the utilization throttle is set to 90% does not mean the utilization is not 90%; it is because many nodes are not fully assigned, yet fewer nodes are completely empty. In addition, in terms of the different algorithms, that the resulting scale-down instance ratios are different is because these algorithms can aggregate scaling down instances in nature, to different degrees, thus putting non-scale-down instances to retries. In conclusion, from these tables, we can conclude that the more segments, the better performance regarding the freed-up node count. And in general, the "Node#-equivalent partitioning" is the best one in practice.

5. RELATED WORK

There are lots of works on data center architectures and service allocation algorithms. [1] proposes a MAC protocol based on TDMA for high bandwidth for MapReduce shuffle workloads. [2] makes the data

centers wireless using the 60GHz technique for augmenting data center networks. [3] explores the potential of using Bloom-Filter routing to enable content discovery in vehicle cloud. [4] poses the 3D signal bouncing solution for inter-rack communication in data centers. [5] discusses the feasibility of completely wireless data centers theoretically and empirically. [6] presentsCarcel, which is a sensor system assisted by clouds. These are about how to leverage wireless communication to enhance service allocation in data centers and how clouds can help wireless networks from an application perspective.

VMTorrent [7] proposes block periodization profile-based prefetch, on-demand fetch, and decoupling of VM image presentation from underlying data-stream. [8] presents a stochastic model of load balancing and scheduling in clouds. It is worth noting that there is also work [9] on flexible-bandwidth abstractions that reduce the cost to cloud tenants. In addition, on a general level, Wrasse [10] addresses resource allocation for the cloud. And Bazaar [11] is a cloud framework offering a job-centric interface for data analytics applications. We are also looking at re-architecting the cloud storage from a view of accelerating the data center updates and improving energy efficiency. In addition, [12] can deliberately guide the placement of workloads to improve performance by selectively switching workloads off poorly performing machines by placement gaming.

To address networking problems in data centers, [13] combats port blackout and TCP outcast problem. In terms of credibility, Depot [14] is acloud storage architecture with minimal trust. VDN [15] redesigns virtual machine imaging in data centers. To save cost, [16] are achieving this by scaling down instances, using less cache. Also node sleep policies are discussed in [17] for saving power for data centers, but not in a service scaling down perspective. [18] proposesa heuristic for consistent VM migration while meeting constraints on bandwidth and loop-freedom. Sybil-Control [19] presents a novel decentralized scheme for controlling the extent of Sybil attacks, where an adversary subverts system operation by emulating the behavior of multiple distinct nodes. As for flow control, PDQ [20] realizes a flow scheduling protocol halting the contending flows by a shortest job first algorithm.

Data center network architectures have also been talked about as they are tightly related to the data center capacity. Jellyfish [21] is a random graph topologically based data center network more cost efficient than fat-trees, and is of more capacity advantage with more scale. HULL [22] uses Phantom Queues and DCTCP to reduce long tail latencies. SCC [23] automates the selection of cluster storage configuration based on service level agreement, hardware and workloads. Netshare and stochastic Netshare [24] are implemented in a sense that bandwidth is allocated predictably across weighted services. The weights are specified by a manager or at each switch port based on a virtual machine heuristic for isolation. Chronos [25] changes the current status of setting low utilization to rein in latency outliers. ThemisMR [26] is a MapReduce implementation that minimizes I/O operations at nearly the speed of TritonSort's record-setting sort performance, and achieves job-level fault tolerance instead of task-level fault tolerance. BlueSky [27] is a LAN-oriented workstation file system backed by cloud storage, a caching proxy architecture as traditional network file service is replaced with commodity cloud services.

Regarding service allocation management, in [28], Delta-SimRank proposes an efficient algorithm to compute SimRank on MapReduce for static graphs and dynamic graphs. VCRIB [29] places rules on both hypervisors and switches level, to achieve a good trade-off between resource usage and performance. It is an abstraction for all types of data center management rules decoupling rule definition from rule partitioning and placement. [30] studies several key requirements and properties for network allocation in data centers, identifying three main requirements: min-guarantee, proportionality and high utilization, and a set of properties to guide the design of allocation policies in the trade-off space. APLOMB [31] holds outsourcing the vast majority of middleboxes from a typical enterprise network without impacting performance, making scalable, affordable middlebox processing accessible to enterprise network of every size. However, these works have not exclusively looked at the solution for saving energy by re-architecting the data center storage with adaptive service scheduling rules so that nodes that host purely scale-down instances are able to be maximized.

6. CONCLUSION

We use an intuitive method that gathering the instances with similar scale-down probability to maximize the freed-up node count in the one-cluster data center. Performing the theoretical derivations, we get the final equations for solving the partitioning end points and the relative boundary conditions. Then we propose four practical partitioning rules informed by the theoretical derivations to do the freed-up node count optimization. The experiment results show that the "node#-equivalent" partitioning algorithm is the best among the proposed four. In future works, we would like to 1) investigate the average improvement of the partitioning rules through several rounds of service scaling-down,2) their impacts on the OS update

performance in terms of duration, and 3) the upper bound of the performance of scale-down probability partitioning performance, where the job size will fully be taken into consideration.

REFERENCES

[1] B. Vattikonda, *et al.*, "Practical TDMA for datacenter Ethernet," *EuroSys'12*
[2] D. Halperin, *et al.*, "Augmenting data center networks with multi-gigabit wireless links," *SIGCOMM'11*
[3] Y. Yu, *et al.*, "Content Routing In The Vehicle Cloud," *MILCOM'12*
[4] X. Zhou, *et al.*, "Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers," *SIGCOMM'12*
[5] J. Shin, *et al.*, "On the Feasibility of Completely Wireless Datacenters," *ANCS'12*
[6] S. Kumar, *et al.*, "A cloud-assisted design for autonomous driving," *SIGCOMM MCC'12*
[7] J. Reich, *et al.*, "VMTorrent: Scalable P2P Virtual Machine Streaming," *CoNEXT'12*
[8] S. Maguluri, *et al.*, "chastic Models of Load Balancing and Scheduling in Cloud Computing Clusters," *INFOCOM'12*
[9] D. Xie, *et al.*, "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," *SIGCOMM'12*
[10] A. Rai, *et al.*, "Generalizing Resource Allocation for the Cloud," *SOCC'12*
[11] V. Jalaparti, *et al.*, "Bridging the Tenant-Provider Gap in Cloud Services," *SOCC'12*
[12] B. Farley, *et al.*, "More for Your Money: Exploiting Performance Heterogeneity in Public Clouds," *SOCC'12*
[13] P. Prakash, *et al.*, "The TCP Outcast Problem: Exposing Unfairness in Data Center Networks," *NSDI'12*
[14] P. Mahajan, *et al.*, "Depot: Cloud Storage with Minimal Trust," *ToCS'11*
[15] C. Peng, *et al.*, "VDN: Virtual Machine Image Distribution Network for Cloud Data Centers," *INFOCOM'12*
[16] T. Zhu, *et al.*, "Saving Cash by Using Less Cache," *HotCloud'12*
[17] A. Gandhi, *et al.*, "Are Sleep States Effective in Data Centers," *IGCC'12*
[18] S. Ghorbani , *et al.*, "Walk the Line: Consistent Network Updates with Bandwidth Guarantees," *HotSDN'12*
[19] F. Li, *et al.*, "SybilControl: practical sybil defense with computational puzzles," *STC'12*
[20] C. Hong, *et al.*, "Finishing flows quickly with preemptive scheduling," *SIGCOMM'12*
[21] A. Singla, *et al.*, "Jellyfish: Networking Data Centers Randomly," *NSDI'12*
[22] A. Alizadeh, *et al.*, "Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center," *NSDI'12*
[23] H. Madhyastha, *et al.*, "scc: Cluster Storage Provisioning Informed by Application Characteristics and SLAs," *FAST'12*
[24] V. Lam, *et al.*, "Netshare and stochastic netshare: predictable bandwidth allocation for data centers," *CCR'12*
[25] R. Kapoor, *et al.*, "Chronos: Predictable Low Latency for Data Center Applications," *SOCC'12*
[26] A. Rasmussen, *et al.*, "Themis: An I/O Efficient MapReduce," *SOCC'12*
[27] M. Vrable, *et al.*, "BlueSky: A Cloud-Backed File System for the Enterprise," *FAST'12*
[28] L. Cao, *et al.*, "Delta-SimRank Computing on MapReduce," *BigMine'12*
[29] M. Moshref, *et al.*, "vCRIB: Virtualized Rule Management in the Cloud," *HotCloud'12*
[30] L. Popa, *et al.*, "FairCloud: Sharing The Network In Cloud Computing," *SIGCOMM'12*
[31] J. Sherry, *et al.*, "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service," *SIGCOMM'12*