DsDs: Data Store Driven Scheduling of Applications For Energy-Efficient and High-Performance Micro-Clouds System

Frezewd Lemma Tena, Christof Fetzer
Chair of Systems Engineering
Dresden University of Technology
Dresden, Germany
{frezewd_lemma.tena1, christof.fetzer}@tu-dresden.de
Web: http://se.inf.tu-dresden.de

***Abstract*—In this work, we present a data store driven scheduling algorithm(DsDs) that colocates applications and data in a special micro-cloud system. Underlying this micro-cloud system is an energy-efficient, consistent-hash based key/value store based on the Apache Cassandra [1] distributed data store. The objective of the proposed scheduling algorithm is to improve the energy and performance efficiency of the whole system. It bases its application placement decisions on the store's perspective.**

The experiments performed using a PlanetLab real world trace and a CloudSim simulation show that DsDs can improve energy saving by more than 20% on top of the the store's saving and without significantly affecting its performance.

I. Introduction

Over the years, computers have seen dramatic increase both in the number of operations they perform per second and the number of operations they perform per watt. In parallel, computers have also seen a dramatic increase in their use for storage and computation, which has fueled the energy consumption of computing. Consequently, energy consumption has become a major cost factor in the total cost of ownership of a cloud computing center.

Studies indicate that up to 50% of the total cost of data center energy is due to the storage infrastructure alone [2]. This is mainly due to the need of many applications running in the cloud to store, replicate and access big data in a database or key-value store.

In this work, we want to exploit the data replication requirements of applications to minimize the storage infrastructure energy consumption using a data store driven scheduling approach(DsDs).

Our work is based on a energy-efficient variant of Apache Cassandra's cloud data store called PowerCass: In a previous work [3], we have shown how a consistent hash based data store like Apache Cassandra can be made energy-efficient using a simple but elegant strategy. Shortly, this strategy divides the storage nodes into three groups: *active, dormant*, and *sleepy*. Nodes in the active group are running continuously, whereas nodes in the dormant and sleepy group are powered on depending on the system load condition. In addition, dormant and sleepy nodes are woken up periodically to synchronize their data with the active nodes. As a result of this simple scheme, PowerCass is able to save up to two-third of the energy consumption during low activity periods.

Now, we want to further improve the energy saving by extending PowerCass to provide a data store driven application scheduling algorithm that co-locates applications and data in a state-of-the art micro-clouds system.

The micro-clouds system we are considering are similar to the Microsoft's data furnace [4]. The main characteristic of this system is that the waste heat from the computing infrastructure is reused in the surrounding building instead of releasing it into the atmosphere.

In this paper, we make the following contributions. Section II describes the system model, including the data store model, the cloud model and the applications model. Section III details the problem space. In Section IV, we give a detailed account of the DsDs algorithm. Section V evaluates DsDs and Section VI put DsDs in to the context of clusters schedulers related works. Finally, we draw our conclusions in Section VII.

II. System Model

This section describes the cloud model, the storage model, the applications model and the energy consumption model of the system that we use in this research.

A. The Cloud Model

In this work, we consider a special kind of an edge cloud. Edge clouds are deployed at the network edge near to the clients. They provide faster response times for their users compared to the few, large-scale cloud centers harboring hundreds of thousands of servers located in the core of the network.

The distributed edge cloud is consists of several sites of the type indicated in Figure 1. Each site has three main components: the micro-cloud, the heating system and a backup heat source.

A micro-cloud hosts both the applications and the storage clusters using a limited number of a relatively homogeneous servers. Each micro cloud is connected to the internet with a

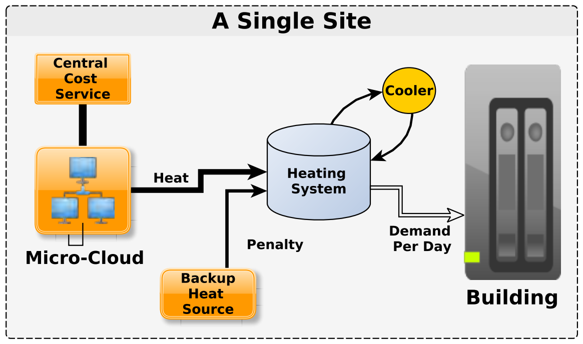

Fig. 1: One Micro-Cloud site.

limited bandwidth and do not share any special inter micro-cloud backbone network. It is also connected to a cost service center that manages the cost of electricity and prices of the system services.

The heating system is responsible for delivering the agreed amount of heat energy to the hosting building. During a short supply of waste heat from the micro-cloud, the heating system draws the required extra energy from the backup heat source. In this case, we say the cloud provider incurs a penalty cost to cover for the extra energy. In the opposite case, when the micro-cloud generates extra heat than is required, the provider also incurs cost to remove the unwanted heat using the attached cooler.

B. The Data Store

As mentioned above, our data store, PowerCass, is a consistent hash based, energy invariant of the highly scalable Apache Cassandra. PowerCass uses the diurnal usage pattern to predict the low, medium and peak activity periods of the system. PowerCass adheres to a specific replication factor: it uses three way replication for writing tasks whereas only one replica for reading tasks. To meet the three way replication for writing tasks during low activity, PowerCass implements a logging scheme.

C. Application Model

In this work, we assume that applications do not have preferable execution sites, meaning applications can not request to be run on a specific micro-cloud. We also assume applications are not tightly coupled in the sense that the output of one application is not the input of another. Moreover, we assume applications resource demand vary considerably, and request demand may go beyond the system capability. We also consider the fact that applications can be either a long running internet services/batch oriented jobs or a short lived small computing tasks. Furthermore, we assume that total demand follows a daily pattern and applications binary sizes are much smaller than the data they require to access.

III. Problem Description

Our goal, given the system model described above, is to minimize the total cost of ownership of a cloud provider mainly by minimizing its energy cost. In such a model, the energy cost of the system can drastically be minimized in two complementary ways: first, if there is demand for heat energy, the cloud provider can reduce its cooling cost – hence its energy cost – by selling the waste heat from a micro-cloud to the surrounding building; second, the provider can save energy by turning off nodes that are idle in a micro-cloud where there is no demand for the waste heat.

To achieve our goal, we need to be able to produce waste heat where it is required by scheduling computations and data accesses at the right micro-cloud. Besides, we also need to be able to avoid moving huge data, ensure load and energy balancing, avoid frequent oscillations of applications, deal with long and short tasks differently, and manage to work seamlessly with the underlying energy-efficient data store to continue having the second way of energy cost minimization.

IV. DsDs Algorithms and Implementations

DsDs is based on running a distributed colocation manager (CM) and a centralized cost service. CM begins working whenever it accepts data access requests with an estimated time of application execution length from clients. Upon receiving the request, CM classifies the requesting application as short or long based on the specified execution length. It similarly classifies the data access tasks. The threshold for the execution length for the applications is determined administratively where as for the data accesses determined by the store. For the classification, CM uses a total of four queues.

Subsequently, CM picks a data access task from one of the two queues while giving a 20% more selection chances for the short tasks queue. This helps to complete the short tasks as quickly as possible while still serving the long, batch or service-oriented tasks. Then CM identifies the storage nodes responsible for the data access task using the store's data partitioner and the replication strategy. The identification of the storage nodes is the most basic information on which the subsequent scheduling activities hang out. Next, CM identifies the current state of the system as low, medium or peak load using the daily pattern. Depending on the state, the store may have powered up only the active nodes, or nodes in the active and dormant nodes, or all the nodes in all the groups, i.e., active, dormant and sleepy groups.

Following the identification of the groups of nodes that are available, and before deciding on which nodes to use for the data access task, CM identifies whether it is a mutation or a reading task. This is because mutation tasks require all the three nodes, whereas reading tasks require only one node. As a result, during low activity periods — where one or two nodes are power downed — mutation tasks necessitate searching to find one or two nodes to perform the write in the temporary logs in order to satisfy the request.

When finally CM decides on which nodes to execute the data access task, it also initiates the co-allocation of the the application in one of the micro-clouds that host nodes. It begins with identifying the cheapest micro-cloud which is described next.

1) Identify Cheap computation site(ICCS) algorithm: In our system, we define cheap computation location as a location where the heat generated by the cloud infrastructure is reused in the surrounding building.

ICCS uses factors such as the electric-cost(e) per killo watt hour(kwh) , the user's heat demand in killowatt per day (D), the application's estimated computation hours, and the penalty cost if the micro-cloud is unable to meet the demand. Therefore, the goal of the algorithm is to meet the required total heat energy in each micro-cloud while avoiding penalties and keeping performance service level agreements(SLA). In other words, we need the following equation hold true in the whole system:

$$\sum E(M_i) \geq D$$

where M_i is micro-cloud i and D is the total demand in the system

$$\sum C(P_i) \leq \sum C(e_i)$$

where P_i is the penalty cost at each micro-cloud i and e_i is the cost of electric city at each micro-cloud i

ICCS begins whenever CM received a read or write task from the clients and initiates it with the nodes and the micro-clouds responsible for the task. Subsequently ICCS consults the Distributed Information Collection(DIC) policy, which is discussed below, to get information about the selected nodes and micro-clouds that includes the current load of each node, and the cost of penalty at each micro-cloud if we do not meet its heat demand. Then ICCS sorts the the list of the nodes in ascending order according to their loads, and the list of micro-clouds in descending order according to their cost.

Once the nodes and the micro-clouds are sorted, ICCS filtered out those micro-clouds generating heat beyond the current demand, and passes the information to the Load and Energy Balancing(LEB) algorithm for system balancing. This left ICCS with the remaining micro-clouds that are candidates for cheap computation location. To determine which of the remaining micro-clouds is the most suitable for the data access as well as the computation, ICSS proceeds as follows: ICCS picks the first elements in both nodes and micro-clouds sorted lists. If the two picks matches, ICCS selects that cloud as the best location for both the data access task and the application. If they don't match, ICCS enters to threshold based comparisons taking into consideration whether a micro-cloud is above or below 95% heat requirements and the node is experiencing above or below 95% load, the detail is listed in Algorithm 1. Moreover ICCS promote and demote nodes if the system is either in low or medium load period.

2) Distributed Information Collection(DIC) Policy and finding a micro-cloud aggregate load: Every second DIC collects metrics on active tasks(actively worked on by the node), pending tasks(queued up in the node), and blocked tasks(blocked due to queue saturation) at each node from the underlying

Algorithm 1 Identify Cheap Computation Micro-cloud

```
1: procedure ICCS($list(nodes, mcs, tt)$)   ▷ receive list of nodes,micro-clouds and the type of the task
2:     for all node n: nodes do
3:         $l \leftarrow dic.load(n)$   ▷ get the current load of each replica from DIC
4:         $loadMap.add(n, l)$
5:         $m \leftarrow microCloudOf(n)$   ▷ get the micro-cloud of node n
6:         $h \leftarrow heat(m)$   ▷ get the current heat production at n's cloud
7:         $c \leftarrow penalty(m))$   ▷ get the cost of penalty from cost service if heat demand is not satisfied
8:         if $h > demand(m)$ then
9:             $heatedMap.add(m, h)$
10:        else
11:            $costMap.add(m, c)$
12:    $sortMap(loadMap, asc)$ ▷ sort nodes based on their load in ascending order
13:    $sortMap(costMap, dsc)$   ▷ sort micro-clouds based on their penalty cost in descending order
14:    $td \leftarrow get_period()$
15:    if $loadMap[0]$ is in $costMap[0]$ then   ▷ check whether the first element in the loads map is in the first micro-cloud in the cost map
16:        if ($td$ is $lowPeriod$ or $midPeriod$) and ($loadMap[0]$ in $dormant$ or $sleepy$ groups) then
17:            $wakeAndPromote(loadMap[0]$
18:        return $costMap[0]$   ▷ return the most cheapest site. This lets CM to add the application to the oscillation list
19:    else   ▷ get time of the day
20:        if $tt$ is $readingTask$ then
21:            $selectedMc \leftarrow nil$
22:            $selectedNode \leftarrow nil$
23:            for all $noden : nodes$ do
24:                for all $mcc : mcs$ do
25:                    if $load(n) < 95\%$ and $heatProduction(c) < 95\%$ then
26:                        $selectedMc \leftarrow c$
27:                        $selectedNode \leftarrow n$
28:            if $selected$ is $nill$ then
29:                $initiateLEB()$   ▷ initiate the load and energy balancing algo
30:            else
31:                if ($td$ is $lowPeriod$ or $midPeriod$) and ($loadMap[0]$ in $dormant$ or $sleepy$ groups) then
32:                    $wakeAndPromote(selectedNode]$
33:                return $selectedMc$
34:        else   ▷ writing task
35:            if ($td$ is $lowPeriod$ or $midPeriod$) then
36:                consider all micro-clouds instead of the three as the reading part do
```

store. DIC is interested only with client initiated tasks. These are the read and mutation tasks which have more priority over the internal tasks such as cache clean up, memtable flushing, compaction, gossip handling, hinted handoff and migration of schema.

Every five second the load information is communicated with the rest of the nodes in the same micro-cloud. When each node collects a minute data from the rest of the nodes in the same micro-cloud, each one compute the exponentially weighted moving average(EWMA) of the queue lengths of the active, pending, and blocked tasks of the the whole micro-cloud. Then the micro-cloud aggregated load immediately communicated to the rest of the micro-clouds by a designated node of that micro-cloud. Together with the load information DIC also disseminates the cost of penalty at each micro-cloud if demand is not going to be met. DIC collects this information from the central cost service.

3) Load and Energy Balancing(LEB) algorithm: LEB algorithm deals with over and under heated clouds as well as over and under loaded clouds and nodes. LEB begins when ICCS initiated it, and ICCS initiates LEB when there is a node that is overloaded in under heated cloud, or a node under loaded in an over heated cloud. To address the former case LEB identifies nodes in the same micro-cloud as the loaded node and iterate over them to find the first node which has a replica peer in an overheated cloud. If such a node is found, LEB terminates the iteration and immediately begins moving applications with long enough execution length to the under heated micro-cloud. To address the latter case LEB tries to move some application to under heated micro-clouds from the overloaded nodes in the micro-cloud. For this purpose LEB identify those applications that are accessing data and have a replica in the micro-cloud of the initiating node. Applications which are eligible according to the oscillation prevention algorithm and have the largest number of blocked and pending tasks will be selected.

In a very low load period, when there are no enough tasks in the system to meet the demand of every micro-cloud heat demand, LEB chooses between two options: lets the system draw electricity from the backup source or execute dummy or non-user tasks. LEB decides between the two based on the cost of penalty according to the cost service and the energy cost to execute the dummy tasks.

4) Oscillation Prevention(OP): To avoid having oscillation of applications among a group of micro-clouds without being executed, some time limit should be reserved before moving a recently scheduled application to a different place. To achieve this we keep applications that have been recently scheduled in an oscillation list, and applications in this list are illegible for rescheduling. Applications leave the oscillation list only after 5 minutes. This keeps the list not to grow indefinitely and also let the system reuse these applications to meet the heat demand wherever they are appropriate.

V. Evaluation

In this section, we evaluate how well DsDs performs. The evaluation goals are the following:

- how does the system's energy consumption improves due to the co-allocation mechanism on top of the data stores on/off strategy?
- Does the system manage to meet the SLA targets at each micro-cloud?
- Is there any performance degradation on the data store due to the addition of the scheduling algorithm?
- How is the application placement time changes as the number of applications to be scheduled increases?

A. Evaluation

To evaluate the proposed algorithm, we have used the CloudSim simulation tool. In evaluating cloud level algorithms, we need to have a large scale, virtualized infrastructure which we can control for repeatable experiments.

In our experiments, we have created up to sixty six power aware micro-clouds, and each one of them hosts upto 12 nodes. As we have indicated in our models, we deal with small and distributed data centers that host a handful of homogeneous nodes. For the experiments we use the HP ProLiant ML110 G4(2.6Ghz cup, 4GB ram, 1GB network bandwidth) servers whose power consumption data is available from spec.org [5] and already modeled in CloudSim [6]. We have used 6 nodes for the data storage and 6 nodes as a computing nodes. Each storage node has peers in two other micro-clouds to simulate the three way replica we have in PowerCass. This enables us to simulate a total of 132 partitions on the consistent hash ring. To simulate the low, medium and peak load times we run each experiment in an 24 hour simulation time. The 24 hour simulation time is divided into 4 slots: from 9-17 as peak, 7-9 and 17-22 as medium, and from 22-6 as low load periods. We vary task execution time randomly from 5 simulation second to represent short tasks to the entire simulation time to represent long running tasks. We use data sizes that randomly vary from 1KB to 2GB. Applications instructions sizes vary for short applications upto fifty thousands, for medium upto hundred fifty thousands and for big upto three hundred thousands.

In the simulation, we run DsDs real code and use a real world trace data from CoMon project which monitors the infrastructure of PlanetLab. This is a data on the CPU utilization for VMs running in 500 different places around the world. This same data is used in the work of [6].

To evaluate the performance efficiency of DsDs, we employed the time the system takes to schedule applications in the appropriate micro-cloud, and the response time of the data store. For the energy saving metrics, we used SLA violation of heat requirements at all the micro-clouds. In addition, we compared how much more energy is saved when we use DsDs co-allocation mechanism on top of the store's original on/off strategy.

B. Performance Evaluation of DsDs

We have conducted two types of experiments to find out the performance of the store after we implemented the scheduling algorithm. First, we run the store with out the scheduling algorithm and collect its latency as the system progresses from low

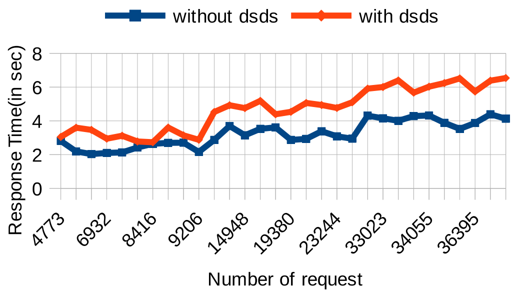

Fig. 2: With and Without DsDs allocation

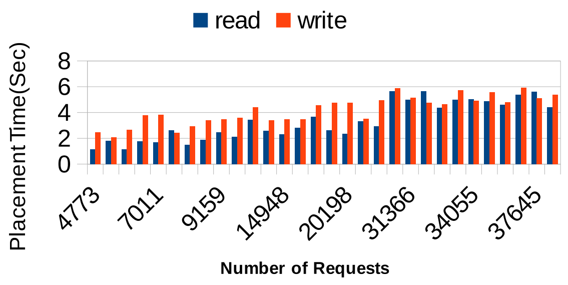

Fig. 3: Applications Placement Time

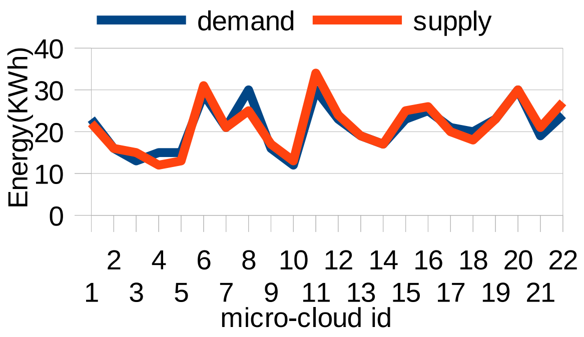

Fig. 4: SLA: demand vs supply of energy across all clouds

to peak load period. Second, we run the same experiment with the scheduling algorithm and see how the latency changes. Figure 2 displays the result of the experiments. As we see from the graph the DsDs algorithm hasn't caused a significant performance degradation to the data store, an increase of 0-2 simulation seconds. Moreover, this small degradation in response time is highly compensated by the energy saving. In the graph, we also observe that some of the highest number of requests get a shorter response time compared to the lower number of requests. This is mainly due to the number of nodes involved in the request coordination at the given time. In the experiment, we randomly select a node to which we send the request for coordination.

1) Average Applications Placement Time : The next logical question to ask is how applications placement time affected as the system progress from low to the peak periods like in the performance case above?

As we mentioned in the algorithm sections, writing and reading tasks have different effects. Accordingly, we run two types of experiments: the first one shows how DsDs placement time changes when only the number of reading tasks increases with the load periods; the second deals with the writing tasks only. Figure 3 shows the result of the experiments.

From the figure we make three observations. The first observation is that placement based on writing requests, in general, takes more time during low and medium load periods, in average 1.24 simulation seconds. This increase is due to the search we make to find the best nodes for temporal logging to compensate for the sleeping nodes during low and medium load periods. For the read-request oriented applications scheduling, we always consider only the three micro-clouds that hosts the three responsible replica nodes. As a result the placement time is totally dependent on the number of requests than the system's different load periods. The second observation is that as the number of requests increases the time the algorithm takes to place application increases. However, the increase in the placement time is not a linear increase with the number of requests. This is due to the uneven distribution of requests coordination by the data store nodes. When we have even distributions for request coordination, the increase in the placement times is linear and becomes smaller which give us a scalable scheduler. The final observation is that the placement time for both read and mutation oriented scheduling becomes similar when the system enters into the peak period. This is an expected behavior because at this period all the responsible nodes for the mutation tasks are awake and available. Consequently the scheduler doesn't need to look for a place for logs, and its activities becomes same for both the read and write tasks.

C. Energy Consumption Evaluation

Here we show how DsDs helps a cloud provider to avoid SLA violation of heat demands.

We make two experiments for this evaluation too: one of the experiments deals with whether an SLA violation exists across all the micro-clouds when we run it at a randomly selected hour of the day; the second experiment, on the other hand, deals with whether an SLA violation exists at a randomly selected micro-cloud through out the day. Figure 4,5, and 6 displays the results of this investigation.

1) SLA violation : The experiments were repeated several times while changing the period for the first experiment and changing the micro-cloud for the second experiment to represent all load cases. We have observed that, as depicted in the figures, DsDs meets its SLA more than 99.9% of the time. The 0.1% is attributed to many short tasks assigned to a cloud and finish long before the load and energy balancing algorithm is initiated.

2) Energy saving: Here we want to see how much more energy saving can be achieved by PowerCass through the ap-

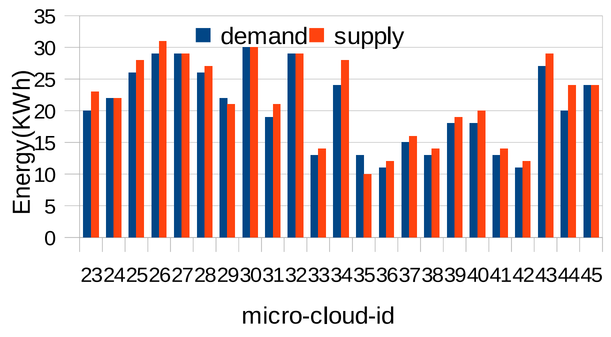

Fig. 5: SLA: demand vs supply of energy across all clouds

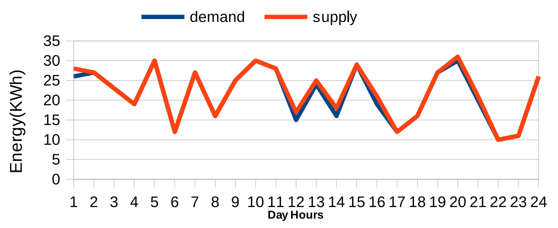

Fig. 6: SLA: One micro-cloud One Day Demand and Supply

plications and data co-allocation scheduling. We have selected a low load and a low heat demand period for the experiment. This is because for the other scenarios we get a clear energy saving advantage. For example, when we have a peak load and a high demand for heat, the saving is obvious as we consume energy only to serve user requests, not for idle execution; and the heat produced is clearly reused to satisfy the high heat demand. But if the system faces a low load and low heat demand period, the store turns on only one-third of the nodes and they should be in the micro-clouds where there are the heat demands for which we need the intelligence of the scheduler. The result of this experiment is displayed in figure 7. The figure display the total energy consumption in the y-axis and the hours of the day in the x-axis. Those hours represents the minimum utilization periods. This figure compares the energy saving with and without DsDs. Accordingly DsDs manages to save at least an additional 20% of the energy consumption which push the total saving to more than 85%.

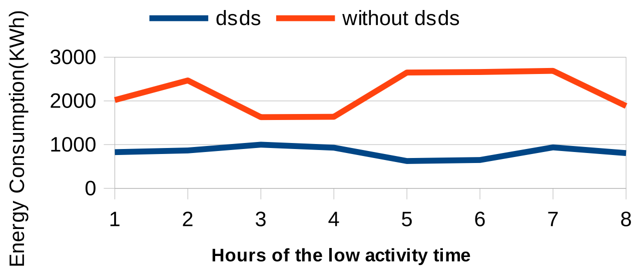

Fig. 7: Only turn on/off vs on/off with co-allocation

VI. Related Work

DsDs, is a hybrid, both data and energy aware application scheduler. In contrast to other data and energy aware scheduling solutions, DsDs takes the data awareness a step further by mainly basing its decisions on the perspective of the data store alone.

Close-to-Files(CF) [7] addresses the issue of data and processors co-allocation in multicluster systems. Using comprehensive search on all combination of data and processors, it allocates tasks to the best fit cluster. CF does not address energy awareness and how busy the file being accessed. DEES [8] is a distributed, energy and data aware solution for a data grid system. It tries to minimize energy consumption by minimizing data replication and task transfers.

EC2 scheduler deals with a coarse grained VM scheduling that give customers start and stop options.

Mesos [9] and Quincy [10] and Firmament [11] are, unlike DsDs, cluster schedulers with a centralized approach which does not scale well with a cloud-level infrastructure.

Omega [12], Apollo [13] and Sparrow [14] similar to DsDs approaches the problem with a distributed and loosely coordinated architecture, but they do not take into considerations energy efficiency of the system.

Eagle [15] is a hybrid scheduler that divides data center nodes into those that handle short jobs versus those that handles long jobs.

Sierra [16], Rabbit [17], SRCMap [18] similarly to our data store try to save energy with on/off mechanism but do not address co-allocation.

VII. Conclusion

In this paper, we have presented DsDs: a distributed, both data and energy aware applications scheduler for a micro-clouds system. Unlike many other cloud schedulers, DsDs takes the data awareness a step further and based its decisions mainly on the perspective of the data store about data access patterns of client applications. Moreover, DsDs fully considers where heat is being reused to improve the systems energy saving capabilities. We have shown through experiments how promising is data store driven scheduling approach for most cloud applications that have data access need from a database or key/value store.

References

[1] A. Lakshman and P. Malik, “Cassandra: A decentralized structured storage system,” *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[2] J. Guerra, W. Belluomini, J. Glider, K. Gupta, and H. Pucha, “Energy proportionality for storage: Impact and feasibility,” *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 35–39, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1740390.1740399

[3] F. L. Tena, T. Knauth, and C. Fetzer, “Powercast: Energy efficient, consistent hashing based storage for micro clouds based infrastructure,” in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 48–55.

[4] J. Liu, M. Goraczko, S. James, C. Belady, J. Lu, and K. Whitehouse, “The data furnace: Heating up with cloud computing,” in *3rd USENIX Workshop on Hot Topics in Cloud Computing*. USENIX, June 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/the-data-furnace-heating-up-with-cloud-computing/

[5] spec.org. (2011) Hewlett-packard company proliant ml110 g3. [Online]. Available: http://www.spec.org/power_ssj2008/results/res2011q1/power_ssj2008-20110127-00342.html

[6] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurr. Comput. : Pract. Exper.*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.1867

[7] H. H. Mohamed and D. H. J. Epema, "An evaluation of the close-to-files processor and data co-allocation policy in multiclusters," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, ser. CLUSTER '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 287–298. [Online]. Available: http://dl.acm.org/citation.cfm?id=1111682.1111730

[8] C. Liu, X. Qin, S. Kulkarni, C. Wang, S. Li, A. Manzanares, and S. Baskiyar, "Distributed energy-efficient scheduling for data intensive applications with deadline constraints on data grids," in *2008 IEEE International Performance, Computing and Communications Conference*, Dec 2008, pp. 26–33.

[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 261–276. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629601

[11] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: fast, centralized cluster scheduling at scale," in *Proceedings of OSDI16: 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, p. 99.

[12] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *SIGOPS European Conference on Computer Systems (EuroSys)*, Prague, Czech Republic, 2013, pp. 351–364. [Online]. Available: http://eurosys2013.tudos.org/wp-content/uploads/2013/paper/Schwarzkopf.pdf

[13] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, 2014, pp. 285–300. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/boutin

[14] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: ACM, 2013, pp. 69–84. [Online]. Available: http://doi.acm.org/10.1145/2517349.2522716

[15] P. Delgado, D. Didona, F. Dinu, and W. Zwaenepoel, "Job-aware scheduling in eagle: Divide and stick to your probes," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, ser. SoCC '16. New York, NY, USA: ACM, 2016, pp. 497–509. [Online]. Available: http://doi.acm.org/10.1145/2987550.2987563

[16] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra : Practical Power-proportionality for Data Center Storage," pp. 169–182, 2011.

[17] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. a. Kozuch, and K. Schwan, "Robust and flexible power-proportional storage," *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, p. 217, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1807128.1807164

[18] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "SRCMap : Energy Proportional Storage using Dynamic Consolidation," *Energy*, no. VM, p. 20, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855531