Energy-aware Hierarchical Scheduling of Applications in Large Scale Data Centers

Gaojin Wen[1] Jue Hong[1] Chengzhong Xu[2] Pavan Balaji[3] Shengzhong Feng[1] Pingchuang Jiang[1]
[1]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
518055 Shenzhen, China
Email: {gj.wen, sz.feng, jue.hong, pc.jiang}@siat.ac.cn
[2]Department of Electricaland Computer Engineering, Wayne State University
Email: czxu@wayne.edu
[3] Mathematics and Computer Science, Argonne National Laboratory
Email: balaji@mcs.anl.gov

***Abstract*—With the rapid advance of cloud computing, large scale data center plays a key role in cloud computing. Energy consumption of such distributed systems has become a prominent problem and received much attention. Among existing energy-saving methods, application scheduling can reduce energy consumption by replacing and consolidating applications to decrease the number of running servers. However, most application scheduling approaches did not consider the energy cost on network devices, which is also a big portion of power consumption in large data centers.**

In this paper we propose a Hierarchical Scheduling Algorithm for applications, namely HSA, to minimize the energy consumption of both servers and network devices. In HSA, a Dynamic Maximum Node Sorting (DMNS) method is developed to optimize the application placement on servers connected to a common switch. Hierarchical crossing-switch adjustment is applied to further reduce the number of running servers. As a result, both the number of running servers and the amount of data transfer can be greatly reduced. The time complexity of HSA is $\Theta(n * log(logn))$, where n is the total number of the severs in the data center. Its stability is verified via simulations. Experiments show that the performance of HSA outperforms existing algorithms.

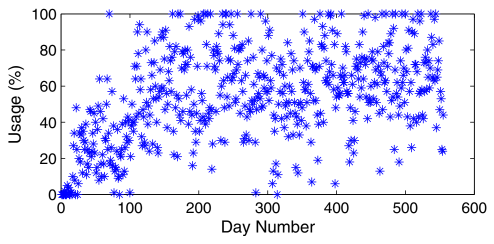
I. Introduction

Modern high-end computing systems including scientific computing centers, cloud computing platforms and enterprise computing systems use tens of thousands of nodes with hundreds of thousands of cores. The upcoming 10-20 petaflop machines such as the Livermore Sequoia, the Argonne Mira supercomputers in the U.S., the Kei supercomputer in Japan, would further increase this number to millions of cores. Finally, with the upcoming exascale platforms, this number is expected to further grow to millions of nodes with close to a billion cores. Such massive-scale systems consume a large amount of energy for operation and cooling. For example, the 2.98 petaflop Dawning Nebula system requires close to 2.55 MW of power to operate. The increasing trend of power consumption would keep growing, if there is no investment into energy and power-awareness research for such systems.

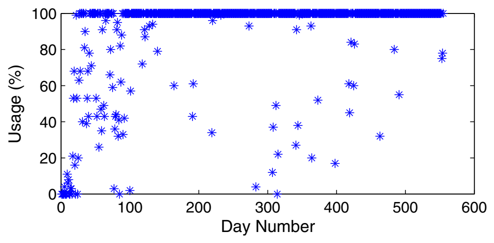
As an effective energy conserving method, application scheduling becomes widely used. By consolidating application on as less servers as possible and making idle servers sleep or power-off, the energy consumption can be significantly reduced[1], [2], [3], [4]. However, few of existing application scheduling approach has considered optimizing the power consumption of the network infrastructure. In a typical commodity large-scale systems, the network infrastructure is built to provide high bisection bandwidth for applications that span the entire system. However, sharing by diverse applications and each utilizing a part of the system, many of these network links are unused. A network usage record from the Fusion cluster at the Argonne National Laboratory of the U.S., spreading over around 18 months from November 2009 to May 2011, testifies such phenomenon (Fig. 1). As shown by this record, the network utilization can vary significantly depending on the forwarding policy (in this case ranging from an average to 54.7% to an average of 90.3%). This instance implies that, by considering the characteristics of the network infrastructure in application scheduling, more energy conservation can be achieved.

In this paper we focus on analyzing application scheduling and its impact on the energy usage of the network infrastructure. We propose a Hierarchical Scheduling Algorithm for applications, namely HSA, to minimize the energy consumption of both servers and network devices. We first present a formal description of the application scheduling problem considering network devices, and define a series of metrics for the evaluation of the algorithm performance. Then, we employ special subset-by-subset and level-by-level scheduling methods to design our HSA algorithm. The proposed HSA algorithm has a time complexity of $\Theta(n * log(logn))$, where n is the total number of the severs in the data center. Through simulation and experiments in a large scale data center, the efficiency and the stability of our algorithm are verified.

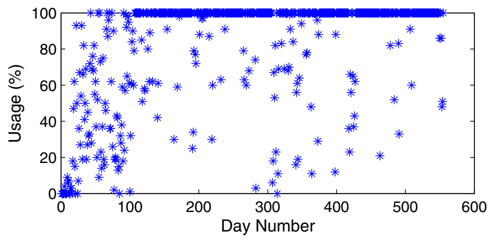
The remainder of this paper is organized as follows. The related work is presented in section II. Problem statement and evaluation metric are formulated in Section III. The main framework of our method is described in Section IV. Scheduling method for node subset is investigated in Section V. The total hierarchy scheduling method is proposed and analyzed in Section VI. We validate it through simulation in a large scale data center in Section VII. Conclusions and discussions on future works are presented in Section VIII.



(a) Minimize energy usage causing as many links as possible to be shared. The average network utilization is 54.7%.)



(b) Maximize performance and assuming that the communication pattern of the applications matches the setup perfectly. The average network utilization is 90.3%.



(c) Use the default linear forwarding table policy of the subnet management infrastructure used by InfiniBand: OpenSM. The average network utilization is 83.3%.

Fig. 1: Fusion network usage with different forwarding policy.

II. Related Work

Workload placement on servers in clustered environments has been traditionally driven by performance objectives. All servers are intended to be fully operational in order to provide the maximum available resources to users; the technique of load balancing is used to spread the workload to as many of them as possible. This mode of operation causes low utilization of the cluster resources, and consequently consumes a high amount of power. Research has pointed out that using techniques of load-screw scheduling and dynamically adjusting idle servers to power-save mode can significantly reduce the energy consumption [11], [1], [2].

Load-screw scheduling is studied extensively in the confront of online bin-packing problem [3], [5], [6], [7]. Each server is treated as a bin with dimensions relative to available resources such as CPU, memory, and network bandwidth. Tasks are treated as objects with their resources requirements as dimensions. A scheduling algorithm determines the minimum number of servers to run these tasks concurrently. Live task migration among servers after scheduling can help to achieve a higher resource utilization rate. It is also important for energy-aware application consolidation, which is another popular technique in green computing [8].

Migration cost is essential to the discussion. Different methods for estimating the cost have been proposed and studied. Gambosi, Sanders,et al. [7] assumed the total number and dimension of the migrated tasks to be the cost in the framework of semi-online bin-packing problem. Lefevre and Orgerie measured energy cost of virtual machines between two different servers [9]. Verma et al. provided a systematical way for migration cost aware virtualized system optimizations [4], [12]. In this paper, we extend these work by considering a more comprehensive and realistic cost model. We include variations in migration cost due to the hierarchical network topology in the data center. Our proposed energy-aware hierarchical scheduling algorithm addresses such variations efficiently.

There has also been a lot of prior work on other models of scheduling including QoS aware job scheduling [13], [14], [15]. The ability of such scheduling techniques to delay the execution of jobs or slow down jobs (as long as the QoS requirement is satisfied) provides even more opportunity for energy savings. Our proposed approach is complementary to these techniques and can be used together.

Application scheduling can be formulated as a bin packing problem (BPP), in which a set of items of various sizes need to be packed by using a minimum number of identical bins. BPP is one of the oldest and most well-studied problems in computer science. Numerous methods have been developed for solving BPP in the past few years, including Best Fit (BF), First Fit (FF), Best Fit Descending (BFD), First Fit Descending (FFD) and so on. An excellent survey of the research on BPP is available in [17] . Let OPT be the minimum number of bins required to pack the input set of items. Packing generated by either FF or BF uses no more than $\frac{17}{10}OPT+2$ bins, and that by either FFD or BFD uses no more than $\frac{11}{9}OPT+4$ bins [17].

III. Problem Statement and Metrics

In this section, we will give the statement of the problem first, and propose an evaluation criterion for the scheduling algorithm.

A. Problem Description

Consider a finite sequence of nodes $Nds = (node_1, node_2, \ldots, node_n)$ and a finite sequence or list of applications $A = (a_1, a_2, \ldots, a_m)$, where an application $a_i \in$ A corresponds to the amount of the resources (Memory, CPU, I/O) that used by the i-th application. We assume that $0 < a_i \leq 100$ for all i and the maximal capacity of each node is 100.

Computer nodes in data center are often connected through high speed switches, as shown in Fig.2. The distance and energy of transferring data between two nodes are different. Transferring data between two nodes connected directly to the same switch tends to use less energy than that of the nodes connected indirectly[10].

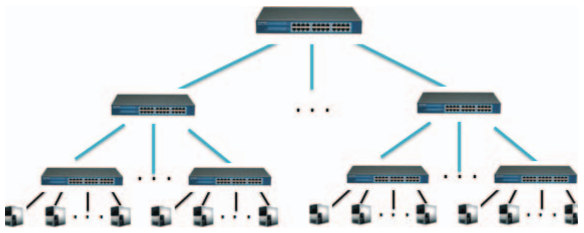


Fig. 2: Network in the data center

The objective of application scheduling is to schedule the applications in A into a subset of nodes in Nds, so as to minimize the maximum number of nodes used and the total size of data transfer incurred during the execution of these applications.

B. State descriptor

Let an integer vector $\mathbf{S^t} = (s_1^t, s_2^t, \dots, s_m^t)$ denote the location of the applications, while $s_i^t \in \mathbf{S}$ means item a_i is located at the node $node_{s_i^t}$ at time t. $\mathbf{S^0}$ represents the initial state of the applications. We conceptually use an integer vector $\mathbf{S} = (s_1, s_2, \dots, s_m)$ to represent a state descriptor, where $1 \leq s_i \leq n$. s_i represents the i-th application will transfer from the its source node $node_{s_i^0}$ to the s_i-th nodes.

C. Objective function

Our objective is to find a new scheduling method to minimize the number of nodes used and data transferred. Two evaluation metrics are considered: node usage metric and the data transfer metric.

1) Node usage metric: , denoted as NU_{cost}, is defined as

$$NU_{cost} = \|\mathbf{s}\|$$

where $\|.\|$ calculates the number of the nodes used by applications A according to the $\mathbf{s}$.

2) Data transfer metric: The cost of transferring data from one node to another are various due to the differences in networks and distance between nodes. Assume a transfer cost weight matrix $\mathbf{C} = \{c_{i,j}\}$, $0 \leq i, j \leq m$, where $c_{i,j}$ is the weight for data transfer from i-th node to j-th node. The data transfer metric, denoted as DT_{cost}, can be defined by the following formula:

$$DT_{cost}(\mathbf{s}) = \sum_{i=1}^{i=m} \|a_i * \sigma(s_i - s_i^0) * c_{s_i, s_i^0}\|$$

where s_i^0 is the node index application a_i resides.

$$\sigma(x) = \begin{cases} 1, & x > 0 \ \ or \ \ x < 0, & \text{(1a)} \\ 0, & x = 0. & \text{(1b)} \end{cases}$$

IV. Algorithm Overview

In this section, we first introduce the concept of node subset and node level, and then present the main framework of our hierarchy scheduling method. It is based on two scheduling strategies: scheduling for node subset and scheduling at a node level.

A. Node subset and node level

We define a node subset to be the set of nodes, in which the cost of data transfer between any two nodes are assumed to be equal, such as the nodes connected to the same switch. Two examples of such a node subset are shown in Fig.3.

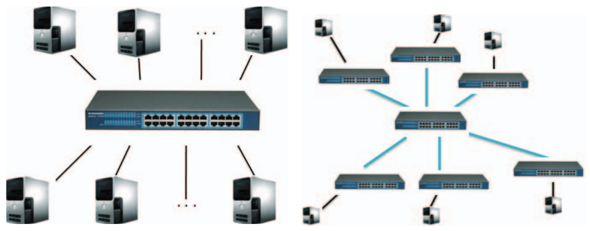


Fig. 3: Two examples of node subset.

A number will be assigned to each subset, as its level mark according to the average cost of the data transfer among the nodes. For example, the level marks of the subsets in Fig.3 are different. Every node level is composed of subsets with the same level mark. Fig.4 shows a description of node levels.

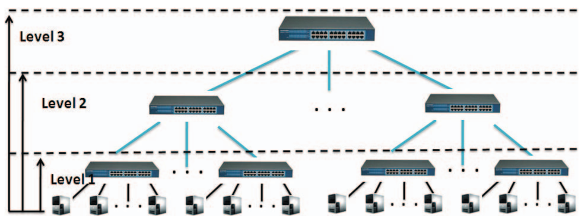


Fig. 4: Examples of node level.

B. Scheduling in node subset

Node subset provides an advantage that nodes in the same node subset can be scheduled without consideration of the cost of the data transfer. Recall that, the scheduling target aims at minimizing the number of nodes used and data transfer. It is obvious that transfer small nodes or applications is superior to larger ones. So nodes in the same node subset with larger or more applications should be kept static and be used to receive applications that transferred from other nodes. In section IV, we will describe the details of the scheduling method within a node subset.

C. Hierarchical Scheduling

After processed by scheduling in node subset, all nodes can be divided into two categories Nd_{full} and Nd_{unfull} according to a preset threshold. Nd_{full} contains the nodes in which the total size of the applications is above the threshold. Nd_{unfull} is the node set of the left nodes which are neither empty nor full and can receive more applications. All nodes in these Nd_{unfull} will be divided into node subsets. For each node subset, application scheduling is done again. Such a hierarchical scheduling process will be executed repetitively. The details are presented in section IV.

V. Scheduling in node subset

For convenience, we use Nd^f denote the nodes in the node subset that are fixed to contain the packages. The left nodes in the node subset is named as transfer nodes set Nd^t.

A. The K-th Max Node Sorting Method (KMNS)

We sort the node sequence according to the amount of the applications that belongs to each node in an ascending order. Then we select the first K largest nodes as Nd^t. The left nodes are Nd^f.

Then we transfer the applications in Nd^t into Nd^f by the Best Fit Descending (BFD) method, since BFD has better performance than other scheduling methods.

During this transfer process, all applications in Nd^t are sorted in descending order, and the applications transfer into the Nd^f from the largest application to the smallest one by the Best Fit method. If an application in Nd^t can not find a holding node in the Nd^f, a new empty node will be added to Nd^f to hold the application.

We shall determine the matching between newly added node in Nd^f and the input node sequence. The newly added node will be matched to the node in the input node sequence from which most applications are transferred.

Finally, we calculate the nodes used and data transfer.

In general, the scheduling method for node subset is as follows:

Algorithm 1 The K-th Max Node Sorting Algorithm

```
Input: one node subset $Nd^s$ with N nodes.
Output: node sequence with the least nodes used and the least data transfer.
Main :
  (1) sort $Nd^s$ according to the amount of the applications of each node in ascend order and generate $Nd^{s'}$.
  (2) $Nd^t = Nd^{s'}_{i=1,...,K}$, $Nd^f = Nds^{s'}_{i=K+1,...,N}$.
  (3) transfer applications from $Nd^t$ into $Nd^f$ with Descending Best Fit method.
  (4) find the matching between $Nd^f$ and $Nd^s$.
  (5) calculate $NU^K_{cost}$ and $DT^K_{cost}$.
```

Property 1: If K=N, then there is no need to sort the input node sequence. The K-th Max Node Sorting Method becomes to be the Best Fitting Descending Method (BFD).

Property 2: $NU^K_{cost} >= N_{opt}$, where the optimal node number N_{opt} can be estimated by the following equations:

$$N_{opt} = \lceil \frac{TA}{ACN} \rceil, \tag{2}$$

where TA is the total size of the applications in the node subset and ACN is the average capacity of the nodes.

Eq.(2) means at least N_{opt} nodes will be used to holding all the applications in the node subset.

Lemma 1: Suppose the total size of the application is T, the optimal node number for the input node sequence is NU^K_{cost}, the average capacity of the node is ACN, the total number of the nodes is N. Then DT^K_{cost} for KMNS is no more than $DTE(K) = T - \frac{NU^K_{cost} * T}{N}$, if $K <= N - NU^K_{cost}$,

Proof: According to the condition, if $K <= N - NU^K_{cost}$, NU^K_{cost} nodes will be used to contain all the applications, the total size of the fixed applications is no less than $NU^K_{cost} * \frac{T}{N}$, so the data transfer is no more than $T - \frac{NU^K_{cost} * T}{N}$. ■

Lemma 1 gives an upper bound to the amount of data transfer for a good scheduling method.

B. Performance of KMNS

To study the performance of the KMNS method, we will use the following three metric:

1) Ratio of Node Used (RoNU):

$$RoNU(K) = NU^K_{cost}/N$$

It is clear that NU^K_{cost} is less than N and bigger than N_{opt}. Consequently, $\frac{N_{opt}}{N} <= RoNU(K) <= 1$.

2) Ratio of Data Transfer (RoDT):

$$RoDT(K) = DT^K_{cost}/T$$

For the data DT^K_{cost} that need to be transferred, the best case is that there is no data, and the worst case is that the data size is $\frac{N-1}{N} * T \longrightarrow T$, $(N \longrightarrow +\infty)$, which is very large. As a result, $0 <= RoDT(K) < 1$.

3) Ratio of Data Transfer Estimation (RoDT):

$$RoDTE(K) = DTE(K)/T = 1 - RoNU(K)$$

From Lemma 1. we know

$$RoDTE(K) >= RoDT(K), \quad if \quad K <= N - NU^K_{cost}$$

With different number of nodes in the input node sequence, we study the performance of KMNS as is shown in Fig. 5 and Fig. 7. It seems that RoNU(K) decreases with respect to K and is no less than N_{opt}/N, while the RoDT(K) increases with respect to K. Transverse K can lead to the best result.

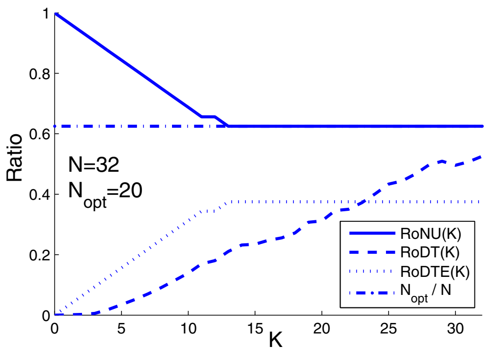


Fig. 5: Performance of KMNS with 32 nodes as input.

Algorithm 2 Dynamic Max Node Sorting Algorithm (DMNS)

```
Input: node sequence $A = \{b_i\}_{i=1}^{N}$,
Output: node sequence $C$ with least nodes used $NU_{cost}$ and least data transfer $DT_{cost}$
Initialization: $NU_{cost} = N$ , $DT_{cost} = +\infty$,$C = null$
Main Loop:
using BFD process all the applications in A, calculate $NU_{cost}^{BFD}$.
for K = 0 to $N$ do
  compute $NU_{cost}^{K}$ , $DT_{cost}^{K}$ and the resulting node sequence B by KMNS algorithm.
  if $NU_{cost} > NU_{cost}^{K}$ then
    $NU_{cost} = NU_{cost}^{K}$, $DT_{cost} = DT_{cost}^{K}$, C=B.
  else if $NU_{cost} == NU_{cost}^{K}$ then
    if $DT_{cost} > DT_{cost}^{K}$ then
      $DT_{cost} = DT_{cost}^{K}$, C=B
    end if
  end if
end for
```

Algorithm 3 Hierarchy Scheduling of Applications (HSA)

```
Input: node sequence $A = \{b_i\}_{i=1}^{N}$, data transfer cost matrix $CM$;
Output: node sequence $C$ with least nodes used and least data transfer
Initialization: current level: $l = 1$, $threshhold = 0.95$,$C = null$
Main Loop:
while $A! = null$ do
  divide A into subsets $\{B_i\}_{i=1}^{N_l}$ according to the transfer cost matrix $CM$
  for i = 1 : $N_l$ do
    process subset $B_i$ with Algorithm 2.
    calculate the $Nd_{unfull}$ and the $Nd_{full}$ according to the $threshhold$
    remove $Nd_{full}$ from A, $C = C \bigcup Nd_{full}$,
  end for
  refine current level : $l = l + 1$;
end while
```

C. *Dynamic Max Node Sorting Method (DMNS)*

Targeting at using least nodes and minimal data transfer, we propose an algorithm of Dynamic Max Node Sorting Method in Algorithm 2.

From property 1, we know that BFD is a special case of DMNS, so $NU_{cost} <= NU_{cost}^{BFD}$. From [17], we know $BFD(L) <= \frac{11}{9}OPT + 4$. The upper bounds for DMNS can be calculated by the following lemma.

Lemma 2:

$$DMNS(L) <= \frac{11}{9}OPT + 4$$

VI. Hierarchy Scheduling

The whole process results in a hierarchy scheduling process as illustrated in *Algorithm 3*.

A. *Complexity*

In whole, the hierarchy scheduling algorithm processes nodes of the same level subset by subset and the nodes level by level. Assume there are n nodes in total and 16 ports in each Switch. The largest level of the hierarchy is $\log_{16} n$. There are n/(16*m) node subsets of level m. Assume the time complexity of the *Algorithm 1* is $\Theta(1)$. so the complexity of the *Algorithm 2* can be calculated by the equation $\Sigma_{m=1}^{\log_{16} n} n/(16 * m) = \Theta(n * log(logn))$.

B. *Stability*

The hierarchy scheduling algorithm process the nodes subset by subset and node subsets level by level. If applications in some node are changed, placement of the applications in the corresponding subset will be refined, which generates a new Nd_{unfull}. So the following hierarchy scheduling from the level 2 to n will be influenced and lead to a new scheduling result. The stability performance of the proposed hierarchy scheduling method can be evaluated by the following Ratio of Local Data Transfer metric:

$$RoLDT = Ns * L/T + DT_{2-n}/T$$

where T is the total size of the application in the input node sequence, L is the maximum capacity of the node, Ns is the number of the nodes in the switch and DT_{2-n} is the total data transfer from the second level to the last level. Different number of the nodes in the switch leads to different level of stability. When the number of the nodes in the Switch Ns increase, the DT_{2-n} will decrease. A good balance between Ns and DT_{2-n} will result in a small $RoLDT$. The smaller $RoLDT$ is, the more stable the hierarchy scheduling algorithm will be and the less the scheduling result will vibrate due to the changes in the applications of the nodes in the node subset.

VII. Experiments and Discussions

The proposed scheduling algorithms has been implemented using C++. We tested the algorithms using PC P-IV, 2.8GHz and 2GB memory. Data transfer weight matrix **C** (as shown in Fig6) is generated according to the distance and network connection among the nodes. Two kinds of input node sequence with size between 0 and 100 are generated randomly with a uniform distribution and a normal distribution (zero mean and a variable standard deviation of 0.3 are used). Then applications satisfying the size constraints of the above pre-generated input nodes are generated randomly with uniform distribution.

A. *Performance of KMNS*

KMNS with 32, 64, 128 and 256 nodes of input node sequence are generated to test the performance of the KMNS. The result is shown in Fig. 5. It seems that RoNU(K) decreases with respect to K and is no less than N_{opt}/N, while the RoDT(K) increases with respect to K.

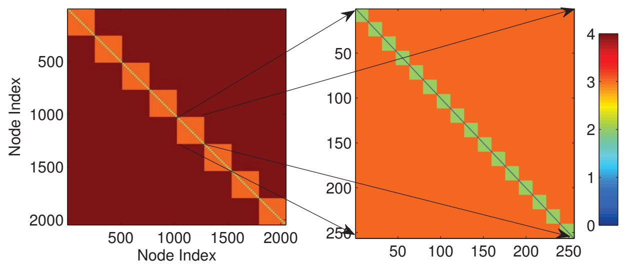


Fig. 6: Transfer cost weight matrix.

B. *Comparison of* DMNS

To compare DMNS, we also have implemented two other scheduling methods: Dynamic Max Application Sorting (DMAS) and Descending Best Fit (BFD). In DMAS, the input node sequence are sorted according to the size of the largest applications in each node. Table I shows the scheduling result of DMNS, DMAS and BFD under the input node sequence with normal distribution. Table II shows the result of input node sequence with uniform distribution. It seems that all three method can achieve the least node used. DMNS is better than DMAS and BFD, which can achieve the least node used together with the minimal data transfer, as is shown in Fig.8.

TABLE I: Comparison for DMNS with input node sequence under normal distribution.

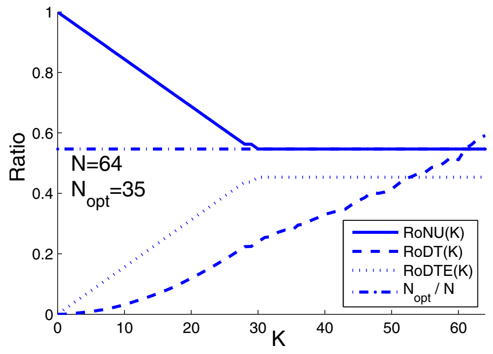
N		DMNS	DMAS	BFD
32	Nd_{cost}	16.58	16.58	16.58
	DT_{cost}	599.77	659.01	1073.85
64	Nd_{cost}	32.26	32.26	32.26
	DT_{cost}	1194.32	1318.99	2176.14
128	Nd_{cost}	63.82	63.82	63.82
	DT_{cost}	2427.37	2712.19	4438.20
256	Nd_{cost}	127.23	127.23	127.23
	DT_{cost}	4853.39	5366.64	8887.95

TABLE II: Comparison for DMNS with input node sequence under uniform distribution.

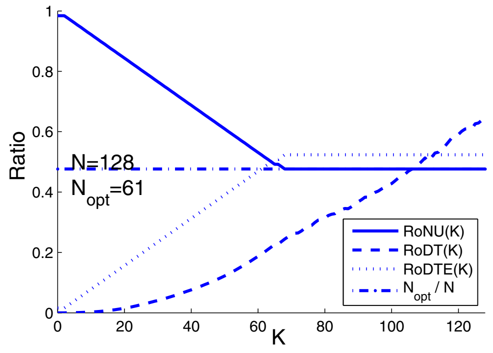
N		DMNS	DMPS	BFD
32	Nd_{cost}	16.16	16.16	16.16
	DT_{cost}	394.46	440.73	932.07
64	Nd_{cost}	31.72	31.72	31.72
	DT_{cost}	806.39	887.54	1909.15
128	Nd_{cost}	63.64	63.64	63.64
	DT_{cost}	1607.06	1800.07	3929.89
256	Nd_{cost}	127.27	127.27	127.27
	DT_{cost}	3237.21	3616.10	7943.63

C. *Performance of* HSA

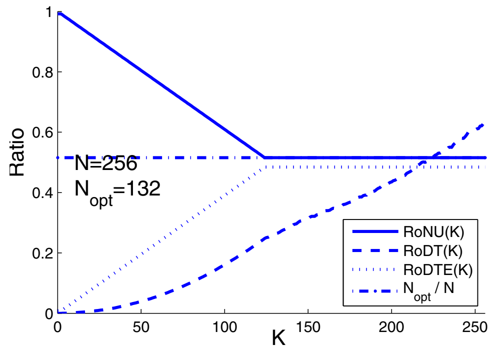
1) Minimal data transfer : We have implemented the hierarchy scheduling algorithm (HSA) proposed in section IV based on DMNS, DMAS and BFD. 4096 nodes generated with normal distribution and uniform distribution are used as input node sequence and tested with 32-, 64-, 128- and 256- port Switch. We find that all three methods can reach the least node used. The difference of the data transfer are compared in Fig.9. In total, DMNS cost the least data transfer among the three methods. It seems that the data transfer cost among the first level is much larger than that of the left levels. Data transfer for normal distribution input is larger than a uniform distribution input. The data transfer of the level 2-n for normal distribution input is almost the same with that for uniform distribution input, when compared to the data transfer of the first level. HSA based on DMNS show the best performance among the three methods, which cost the least data transfer when the number of the nodes in the Switch is 32 and 64. All three method cost almost the same data transfer of the level 2-n, when the number of the nodes in the Switch is 128 and 256. For each scheduling method, the larger the number of nodes in the Switch is, the less data transfer of the level 2-n



(a) With 64 nodes as input.



(b) With 128 nodes as input.



(c) With 256 nodes as input.

Fig. 7: Performance of KMNS.

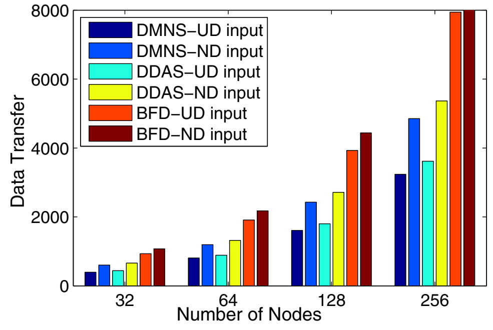


Fig. 8: Comparison of Hierarchy Scheduling Algorithm. The Performance of DMNS, DMAS, BFD is tested with the uniform distribution input (UD) and normal distribution (ND) input.

between the node subset is. In total, the proposed hierarchy scheduling algorithm is a stable scheduling method.

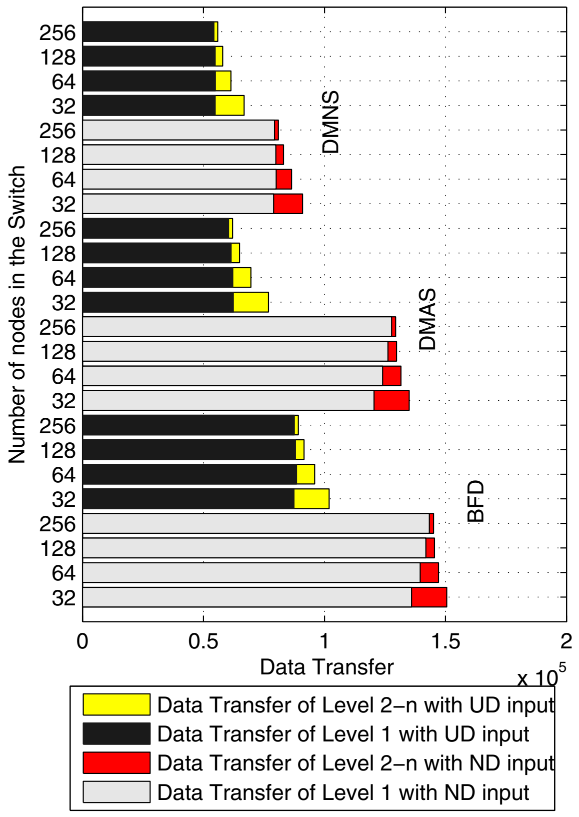


Fig. 9: Comparison of the performance of hierarchy scheduling algorithm (HSA) based on DMNS, DMAS and BFD. HSA based on DMNS shows the best performance in total.

2) Comparison of the stability: We compare the stability of the hierarchy scheduling based on DMNS, DMAS and BFD in Fig.10. It seems that different number of the nodes in the Switch leads to different stability. With 64 nodes in each Switch, all three methods can reach better stability than with 32, 128 or 25 nodes in each Switch. In total HSA based on DMNS shows the better stability than HSA based on DMAS and BFD. With 64 nodes in each Switch, HSA based on DMNS shows the best performance.

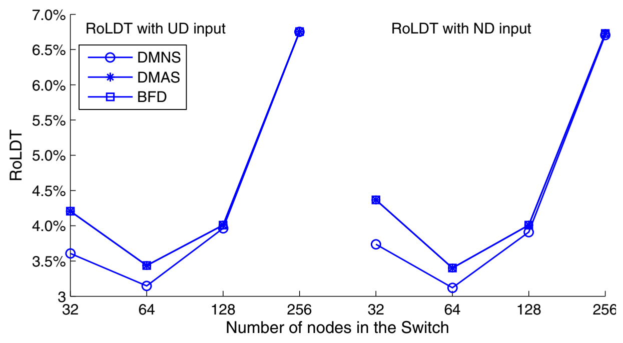


Fig. 10: Comparison of the stability of the hierarchy scheduling based on DMNS, DMAS and BFD. With 64 nodes in each Switch, HSA based on DMNS shows the best performance in total.

VIII. Conclusion and Future Work

In this paper we solved the application scheduling problem in large scale data center. Targeting at using least servers and minimal data transfer, a dynamic maximum node sorting (DMNS) scheduling method is proposed to optimize the application scheduling for the nodes connected by a common switch. Based on DMNS, a stable hierarchy scheduling algorithm (HSA) with time complexity of $\Theta(n * log(logn))$ is developed. The stability of HSA stems from the idea of scheduling applications subset-by-subset and level-by-level. The performance of HSA are compared with similar traditional algorithms via simulation. Results show that HSA outperforms the others in terms of stability and energy consumption.

Future work includes considering more energy-aware factors in our scheduling algorithm. For instance, the cooling conditions of each node are different. The application is preferred to be scheduled to nodes with lower temperature, which leads to the development of a temperature-aware scheduling algorithms for applications in large scale data center.

Acknowledgment

This work is supported by the National High-Tech Research and Development Plan of China (863 program) under Grant No.2009AA01A129-2, the Science and Technology Project of Guangdong Province under the grant No.2010A090100028, 201120510102, National Natural Science Foundation of P. R. China under the Grant No.60903116and Knowledge Innovation Project of the Chinese Academy of Sciences No.KGCX2-YW-131, the Science and Technology Project of Shenzhen under the grant No. JC200903170443A, ZD201006100023A, ZYC201006130310A.

References

[1] Anne-Cecile Orgerie, Laurent Lefevre, Jean-Patrick Gelas, Demystifying energy consumption in Grids and Clouds, GREENCOMP, pp.335-342, International Conference on Green Computing, 2010

[2] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. 2001. Managing energy and server resources in hosting centers. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01). ACM, New York, NY, USA, 103-116.

[3] Andrew Yao. New algorithm for bin-packing. Journal of ACM, **27**,207-227 (1980)

[4] Akshat Verma, Puneet Ahuia, Anindya Neoqi. Power-aware dynamic placement of HPC applications. Proceedings of the 22nd annual international conference on Supercomputing (ICS'08), 175-184.

[5] . David Johnson. Near-Optimal Bin Packing Algorithms. PhD thesis, MIT, Cambridge, Massachusetts, June 1973

[6] G. Gambosi, A. Postiglione, and M. Talamo. Algorithms for the relaxed online bin-packing model.SIAM Journal on Computing, **30(5)**, 1532-1551 (2000)

[7] Peter Sanders, Naveen Sivadasan, and Martin Skutella, Online Scheduling with Bounded Migration, Springer, 2004

[8] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. 2008. Energy aware consolidation for cloud computing. In Proceedings of the 2008 conference on Power aware computing and systems (HotPower'08). USENIX Association, Berkeley, CA, USA, 10-10.

[9] Laurent Lefevre, Anne-Cecile Orgerie. Designing and Evaluating an Energy Efficient Cloud. The Journal of Supercomputing, **51(3)**:352-373, March 2010

[10] Baliga, J.; Ayre, R.W.A.; Hinton, K.; Tucker, R.S.; , Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport, Proceedings of the IEEE , **99(1)**, 149-167 (2011)

[11] Andrew J. Younge, Gregor von Laszewski, Lizhe Wang, Sonia Lopez-Alarcon, and Warren Carithers. 2010. Efficient resource management for Cloud computing environments. In Proceedings of the International Conference on Green Computing (GREENCOMP '10). IEEE Computer Society, Washington, DC, USA, 357-364.

[12] Akshat Verma, Puneet Ahuia, Anindya Neoqi. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, pp. 243-264. 2008.

[13] M. Islam, P. Balaji, P. Sadayappan and D. K. Panda, QoPS: A QoS based scheme for Parallel Job Scheduling. IEEE Springer LNCS Journal Series, **2862**, 252-268 (2003)

[14] M. Islam, P. Balaji, G. Sabin and P. Sadayappan, Analyzing and Minimizing the Impact of Opportunity Cost in QoS-aware Job Scheduling. In the IEEE International Conference on Parallel Processing (ICPP'07), 42.

[15] M. Islam, P. Balaji, P. Sadayappan and D. K. Panda, Towards Provision of Quality of Service Guarantees in Job Scheduling. In the proceedings of the IEEE International Conference on Cluster Computing (Cluster'04), 245-254

[16] N. Desai, P. Balaji, P. Sadayappan and M. Islam. Are Non-Blocking Networks Really Needed for High-End-Computing Workloads?. IEEE International Conference on Cluster Computing (Cluster'08), 152-159.

[17] Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., and Graham, R.L., Worst-case Performance Bounds for Simple One-Dimensional Packing Algorithms, SZAM Journal on Computing, **3(4)**, 299-325 (1974).