IEICE TRANS. INF. & SYST., VOL.E99–D, NO.6 JUNE 2016

LETTER

Dominant Fairness Fairness: Hierarchical Scheduling for Multiple Resources in Heterogeneous Datacenters

Wenzhu WANG[†a)], ***Nonmember***, **Kun JIANG**[†], ***Student Member***, **Yusong TAN**[†], ***and*** **Qingbo WU**[†], ***Nonmembers***

SUMMARY Hierarchical scheduling for multiple resources is partially responsible for the performance achievements in large scale datacenters. However, the latest scheduling technique, Hierarchy Dominant Resource Fairness (H-DRF) [1], has some shortcomings in heterogeneous environments, such as starving certain jobs or unfair resource allocation. This is because a heterogeneous environment brings new challenges. In this paper, we propose a novel scheduling algorithm called Dominant Fairness Fairness (DFF). DFF tries to keep resource allocation fair, avoid job starvation, and improve system resource utilization. We implement DFF in the YARN system, a most commonly used scheduler for large scale clusters. The experimental results show that our proposed algorithm leads to higher resource utilization and better throughput than H-DRF.

key words: *hierarchical scheduling, heterogeneous datacenter, multiple resources, H-DRF*

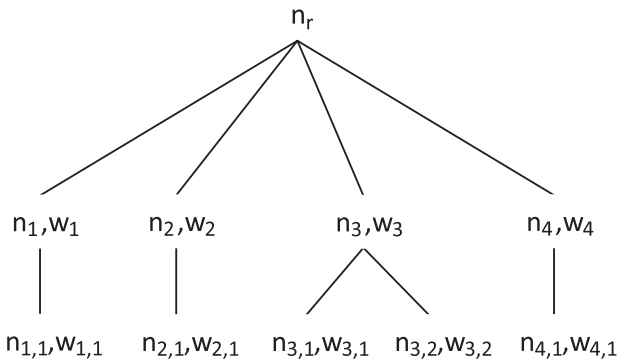

Fig. 1 Example of organizational hierarchy.

1. Introduction

Datacenters are becoming increasingly popular in the big data processing domain. One of its important features is that it provides multiple types of resources, such as CPUs, memory, and coprocessors [2]. Data processing tasks can use several of them to process their data set. Therefore, scheduling these resources efficiently is an important factor for obtaining high performance. For this reason, there has been much research on multiple resource scheduling [3]–[6]. Among them, the most famous one is the Dominant Resource Fairness (DRF) [7] scheduling algorithm, which has already been integrated into the Hadoop YARN [8] system, the most commonly used scheduler for large scale clusters.

Because hierarchical scheduling can reflect organizational priorities, it has been supported by most scheduling systems, such as the Capacity Scheduler [9] and Fair Scheduler [10]. Taking Fig. 1 as an example, a leaf node represents a demanding job, and a non-leaf node represents a department or job queue. Each node i has a weight w_i, which is a basis for obtaining resources, i.e., node n_3 should get $w_3/\sum_{i=1}^{4} w_i$ resources from its parent node n_r, and node $n_{3,1}$ should get $w_{3,1}/\sum_{i=1}^{2} w_{3,i}$ resources from n_3. Because DRF has some shortcomings with respect to hierarchical scheduling, Arka et al. proposed Hierarchy DRF (H-DRF) [1], which can deal with the issues in DRF such as resources left unallocated or the starving of certain jobs.

It is noteworthy that heterogeneity is a newly emerging feature for multiple resources, which means that there are large differences in the number and performance of different resources [11]–[13]. For example, in our real datacenter, Tianhe-2 system [14], the ratio of the CPUs to coprocessors is as high as 16. In addition, the performance between the CPUs and coprocessors also varies considerably [15]. However, hierarchical scheduling in a heterogeneous environment brings new challenges. First, it is difficult to allocate asymmetrical multiple resources fairly based on the dominant resource because the resources with lower amounts have a low allocation priority. Second, unfair resource allocation may result in some resources saturating quickly, so certain jobs, which are only allocated a few resources, may be starved because of the lack of saturated resources. Moreover, unfair resource allocation can also result in low resource utilization. Unfortunately, neither DRF nor H-DRF can handle these issues.

In this paper, we introduce Dominant Fairness Fairness (DFF), a novel hierarchical scheduling algorithm for multiple resources allocation in both heterogeneous and non-heterogeneous clusters. This algorithm attempts to satisfy the resource demands fairly among all jobs and share the unused resources among other nodes with demands. Moreover, DFF can also avoid the job starvation that results from unfair resource allocation. Finally, we implement the DFF algorithm in a Hadoop YARN system, the most commonly used scheduler for large scale clusters, to schedule real diverse workloads.

To evaluate the performance of DFF, we ran five real workloads in a CPU-MIC heterogeneous cluster and scheduled these workloads using both H-DRF and DFF. The experimental results show that DFF outperforms the H-DRF

Manuscript received December 12, 2015.
Manuscript publicized March 3, 2016.
[†]The authors are with College of Computer, National University of Defense Technology, China.
a) E-mail: wenzhuw@gmail.com
DOI: 10.1587/transinf.2015EDL8253

scheduling algorithm in terms of resource utilization and system throughput.

2. H-DRF

Before introducing H-DRF, we present Fig. 1 as an example to introduce some basic concepts. The parent of a node is given by $P()$, i.e., $P(n_1) = n_r$, and the set of the children of node n_i is given by $C(n_i)$. Function $A()$ tasks a set of nodes and returns the subset of those nodes that are currently demanding resources, and $A_j()$ returns the subset that are currently demanding resource j. For example, if node i is demanding resource j, it should get a fraction of resource j from its parent as follows.

$$\frac{w_i}{\sum_{k \in A_j(C(P(n_i)))} w_k} \tag{1}$$

In DRF scheduling, each job has a dominant resource, which is the resource that has the highest share of the system's total resources. DRF seeks to equalize the dominant shares across all jobs. However, using DRF in hierarchical scheduling may starve certain jobs or leave resources unallocated. Therefore, H-DRF generalizes the DRF algorithm to support hierarchical scheduling by rescaling the resource consumption vector of the non-leaf nodes. For example, assume a non-leaf node N has several children demanding resources and child i has the minimum dominant share M_i. For each child j ($j \in A(C(N))$), the resource consumption vector is then multiplied by M_i/M_j. Finally, all the children's rescaled vectors are summed to obtain node N's resource consumption vector.

By doing this, the real shared resource vector of node N is reduced to improve the resource allocation priority. However, this also harms the fairness between all nodes because the real value of the dominant resource has been changed.

3. DFF Scheduling

The objectives of DFF are to keep resource allocation fair, avoid job starvation, and improve system resource utilization. Before introducing DFF, we first define some concepts: *Fair Resource* and *Fairness*. We then propose the dynamic DFF scheduling algorithm.

3.1 Definitions

We first define the concept of a *Fair Resource*, which is the prerequisite for *Fairness*.

Definition 1: Fair Resource: The amount of resources that a node should get in fairness.

Each node has a Fair Resource vector FR, as Formula 2 illustrates, and $fr_{i,j}$ represents the ideal amount of resource j ($1 \le j \le m$) that node i should be allocated in fairness.

$$FR_i =< fr_{i,1}, fr_{i,2}, \ldots, fr_{i,j} > \tag{2}$$

We can compute each $fr_{i,j}$ as follows: If node i does not need resource j, then $fr_{i,j} = 0$. However, if node i needs resource j, $fr_{i,j}$ is calculated according to

$$fr_{i,j} = \frac{fr_{P(n_i),j} * w_i}{\sum_{k \in A_j(C(P(n_i)))} w_k} \tag{3}$$

where $fr_{P(n_i),j}$ is the fr_j of node i's parent $P(n_i)$, w_i is the weight of node i, and set $A_j(C(P(n_i)))$ comprises the sibling nodes of node i that also demand resource j. Using Formula 3, all of resource j belonging to the parent node can be fairly allocated to its children according to their weights. Taking Fig. 1 as an example, n_r is the root node, so $FR_{n_r} = R$, where R is the total system resource vector. Node n_3 should get resource j from n_r as $(fr_{n_r,j} * w_3)/\sum_{k \in A_j(C(n_r))} w_k$.

The Fair Resource of each node should be updated, i.e., reallocated, when a new job is submitted or an old job has been completed. Algorithm 1 illustrates the pseudo-code for updating FR in a hierarchical structure. It is a recursive function, and should be called as UpdateFR(n_r).

Algorithm 1 UpdateFR(n_i) pseudo-code

```
1: if n_i is a leaf node then
2:    do nothing
3: else
4:    for all nodes c (c ∈ C(n_i)) and j (1 ≤ j ≤ m) do
5:       if node c needs resource j then
6:          fr_{c,j} = (fr_{i,j} * weight_c) / Σ_{k∈A_j(C(n_i))} weight_k
7:       else
8:          fr_{c,j} = 0
9:       end if
10:      UpdateFR(n_c)
11:   end for
12: end if
```

Definition 2: Fairness: The *fairness* of node i is the maximum ratio of the resource that it has consumed to the resource that it should get in fairness.

We use $f_{i,j}$ to represent the fairness of resource j ($1 \le j \le m$) of node i. Formula 4 illustrates the vector of node i's fairness vector F_i:

$$F_i =< f_{i,1}, f_{i,2}, \ldots, f_{i,m} > \tag{4}$$

According to the definition of fairness, we can calculate the value of $f_{i,j}$ as Formula 5, where $u_{i,j}$ is the amount of resource j that node i has consumed.

$$f_{i,j} = \frac{u_{i,j}}{fr_{i,j}} \tag{5}$$

Finally, the fairness of node i f_i is calculated as follows:

$$f_i = max_{j=1}^{m}\{u_{i,j}/fr_{i,j}\} \tag{6}$$

Therefore, when $u_{i,j}$ is equal to $fr_{i,j}$, it means that node i has consumed all of the resource j that it should get in fairness.

Resource Sharing: Function $Y()$ takes a set of nodes and returns the subset of those nodes that can be scheduled, i.e., for each node i in $Y()$, $D_i \le R - C$, where D_i is the

Algorithm 2 UpdateF(n_i) pseudo-code

```
if n_i is a leaf node then
    f_i = max_{j=1}^{m} {u_{i,j}/fr_{i,j}}
else
    for all nodes c (c ∈ A(C(n_i))) do
        UpdataF(c)
    end for
    U_i = Σ U_k    (k ∈ C(n_i))
    f_i = max_{j=1}^{m} {u_{i,j}/fr_{i,j}}
    if f_i > 1 && f_j < f_i (f_j = min f in Y(A(C(n_i)))) then
        f_i = f_j
    end if
end if
```

Algorithm 3 Dynamic DFF pseudo-code

```
n_i = n_r
while n_i is not a leaf node do
    n_j = the node with the lowest f_i in Y(A(C(n_i)))
    n_i = n_j
end while
D_i ← demand of node i's next task
C = C + D_i
U_i = U_i + D_i
UpdateF(n_r)
if there is a new submitted job or an old finished job then
    UpdateFR(n_r)
end if
```

demand of node i's next task, and C are the resources that the system has consumed in total. A node is blocked if any of the resources it requires are saturated, i.e., $Y(A(n_i)) = \emptyset$, but $A(n_i) \neq \emptyset$. When node i is blocked, resource $fr_{i,j}$ may not be completely consumed by node i, so the rest of $fr_{i,j}$ can be shared by the other sibling nodes. As a result, node k ($k \in A_j(C(P(n_i)))$) can obtain extra resource j beyond $fr_{k,j}$, i.e., f_k may be larger than one. Note that if f_i of a non-leaf node i is larger than one, we should check whether $f_j < f_i$, where $f_j = \min f$ in $Y(A(C(n_i)))$. This is because when $f_i > 1$, child node j may be blocked, and other sibling nodes of node j may have extra resources. Therefore, when node j is unlocked and $f_j < f_i$, we should make $f_i = f_j$ to reflect any new demands from node j. Algorithm 2 illustrates the pseudo-code for updating F in a hierarchical structure. It is also a recursive function, and should be called as UpdateF(n_r).

3.2 DFF

In this section, we propose the dynamic DFF algorithm based on the concept of fairness. Algorithm 3 shows the pseudo-code for DFF.

First of all, we find the leaf node i with the lowest fairness and $Y(A(n_i)) \neq \emptyset$. By increasing the lowest f_i, DFF can equalize the fairness across all nodes in a cluster. DFF then allocates the demanded resources D_i to node i. Meanwhile, some tasks may have finished and released their resources accordingly. Because the system resource consumption has changed, DFF needs to update the fairness of each node. Finally, if there a new node submitted or an old node finished, the Fair Resource should be updated, because the hierarchical structure of the system has been changed.

4. Evaluation

In this section, we evaluate the performance of DFF compared with H-DRF. We first implement DFF in the YARN scheduler, then compared it with H-DRF in a CPU-MIC heterogeneous cluster. Finally, we show the experimental results.

4.1 Experimental Setup

We deployed a CPU-MIC heterogeneous cluster for diverse workload scheduling. The cluster includes seven computing nodes equipped with 14 CPUs and 21 MICs. The MIC coprocessor is an Intel Xeon Phi 3120P, and the CPU is an Intel Xeon E5-2670. Because each CPU has 24 hardware threads, the total CPU resource is 336 (14×24). Other configurations are as follows: the host memory of each node is 136 G, the hard drive is a 500 G SATA3 magnetic hard disk, and 64-bit RedHat 6.2 was used as the operating system.

We used five real diverse jobs, which are described in Table 1, to evaluate the performance. The hierarchical structure was set as shown in Fig. 1.

4.2 Experimental Results

First, we observe the resource utilization for different algorithms. Figure 2 shows the utilization of CPU, Memory, and

Table 1 Description of the input jobs.

Job ID	Job Node	Resource Demand Vector <CPU, Memory, MIC>	Number of Tasks
1	$n_{1,1}$	<4, 4, 0>	300
2	$n_{2,1}$	<4, 3, 0>	370
3	$n_{3,1}$	<2, 4, 0>	260
4	$n_{3,2}$	<1, 4, 2>	180
5	$n_{4,1}$	<1, 3, 2>	280

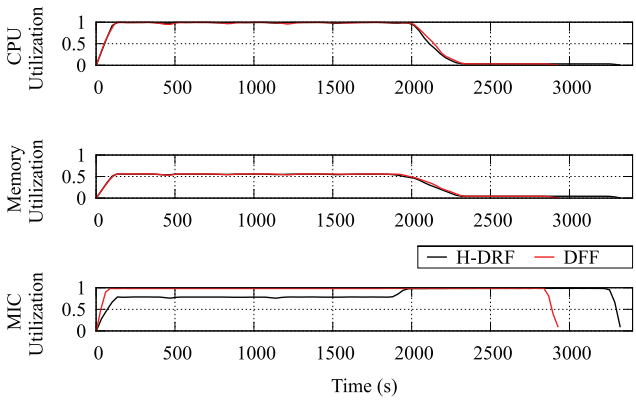

Fig. 2 CPU, Memory, and MIC utilization for H-DRF and DFF for a trace of five diverse workloads.

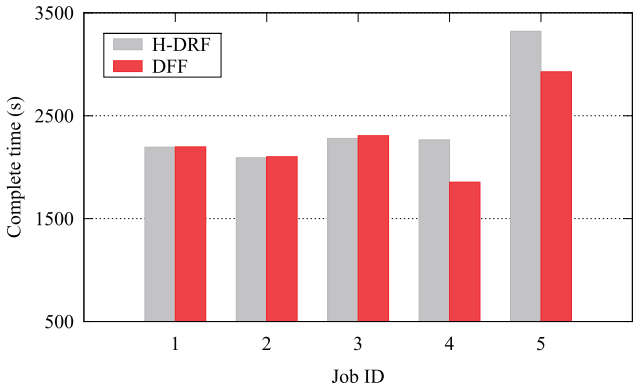

Fig. 3 Completion times for a trace of five diverse jobs using H-DRF and DFF.

MIC while the five jobs ran. We can see that the MIC utilization is improved greatly between 0 and 2,000 s by the DFF algorithm. This is because when using H-DRF, the CPU resource is saturated quickly as a result of unfair resource allocation. When certain jobs demand the MIC resource, there is no CPU to satisfy the demand vector. As a result, MIC resources cannot be allocated, and the jobs demanding the MIC resource will be starved. These jobs can only obtain resources when some CPU resources are released after 2,000 s. In contrast, DFF can maintain fairness among all nodes and allocate resources fairly to avoid job starvation. In addition, DFF only incurs a small amount of overhead because the CPU and Memory utilization do not change much. Overall, DFF can greatly improve resource utilization.

Figure 3 shows the complete times of the five jobs using the H-DRF and DFF algorithm. We can see that the complete times of the MIC-demanding jobs (Jobs 4 and 5) have been reduced greatly. This is because DFF can eliminate the starvation of MIC-demanding jobs by allocating resources fairly and speeding up their processing. Meanwhile, the complete times of the CPU-demanding jobs (Jobs 1, 2, and 3) have not changed very much. This is because DFF does not sacrifice other jobs' performance when increasing some certain jobs' resources. Overall, the total complete time of the five jobs has been reduced by about 11.9% using the DFF algorithm. Therefore, we can see that DFF has a better throughput than H-DRF in a heterogeneous environment.

5. Conclusion

In this paper, we proposed DFF, a hierarchical scheduler for multiple resources in heterogeneous datacenters. We first defined the concept of a Fair Resource and Fairness, which can reflect the fairness among all nodes. DFF attempts to maintain the fairness of resource allocation based on Fairness. The experimental results show that DFF can achieve higher resource utilization and better throughput than the H-DRF sharing scheme in heterogeneous environments.

References

[1] A.A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical scheduling for diverse datacenter workloads," Proceedings of the 4th annual Symposium on Cloud Computing, ACM, 2013.
[2] B. Sharma, R. Prabhakar, S.-H. Lim, M.T. Kandemir, and C.R. Das, "MROrchestrator: A fine-grained resource orchestration framework for MapReduce clusters," 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), pp.1–8, 2012.
[3] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multiresource Allocation: Fairness–Efficiency Tradeoffs in a Unifying Framework," IEEE/ACM Transactions on Networking, vol.21, no.6, pp.1785–1798, 2013.
[4] D.C. Parkes, A.D. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," ACM Transactions on Economics and Computation, vol.3, no.1, pp.1–22, 2015.
[5] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," ACM SIGCOMM Computer Communication Review., vol.42, no.4, pp.1–12, 2012.
[6] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," Proceedings of the 8th ACM European Conference on Computer Systems, pp.351–364, 2013.
[7] A. Ghodsi, M. Zaharia, B. Hindman, et al., "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," NSDI, p.24, 2011.
[8] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet another resource negotiator," Proceedings of the 4th annual Symposium on Cloud Computing, pp.1–16, 2013.
[9] Hadoop Capacity Scheduler. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html
[10] Hadoop Fair Scheduler. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html
[11] R. Farivar, A. Verma, E.M. Chan, and R.H. Campbell, "Mithra: Multiple data independent tasks on a heterogeneous resource architecture," IEEE International Conference on Cluster Computing and Workshops, 2009. CLUSTER '09, pp.1–10, 2009.
[12] J.A. Stuart and J.D. Owens, "Multi-GPU MapReduce on GPU clusters," 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS), pp.1068–1079, 2011.
[13] M. Lu, Y. Liang, H.P. Huynh, Z. Ong, B. He, and R.S.M. Goh, "MrPhi: An optimized MapReduce framework on Intel Xeon Phi coprocessors," IEEE Transactions on Parallel and Distributed Systems, vol.26, no.11, pp.3066–3078, 2014.
[14] http://www.top500.org/lists/2015/06/
[15] G. Teodoro, T. Kurc, J. Kong, L. Cooper, and J. Saltz, "Comparative Performance Analysis of Intel (R) Xeon Phi (TM), GPU, and CPU: A Case Study from Microscopy Image Analysis," 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp.1063–1072, 2014.