Bandwidth-Aware Scheduling With SDN in Hadoop: A New Trend for Big Data

Peng Qin, Bin Dai, Benxiong Huang, and Guan Xu

***Abstract*—Software-defined networking (SDN) is a revolutionary network architecture that separates out network control functions from the underlying equipment and is an increasing trend to help enterprises build more manageable data centers where big data processing emerges as an important part of applications. To concurrently process large-scale data, MapReduce with an open-source implementation named Hadoop is proposed. In practical Hadoop systems, one kind of issue that vitally impacts the overall performance is known as the NP-complete minimum make span[1] problem. One main solution is to assign tasks on data local nodes to avoid link occupation since network bandwidth is a scarce resource. Many methodologies for enhancing data locality are proposed such as the Hadoop default scheduler (HDS) and state-of-the-art scheduler balance-reduce scheduler (BAR). However, all of them either ignore allocating tasks in a global view or disregard available bandwidth as the basis for scheduling. In this paper, we propose a heuristic bandwidth-aware task scheduler bandwidth-aware scheduling with SDN in Hadoop (BASS) to combine Hadoop with SDN. It is not only able to guarantee data locality in a global view but also can efficiently assign tasks in an optimized way. Both examples and experiments demonstrate that BASS has the best performance in terms of job completion time. To our knowledge, BASS is the first to exploit talent of SDN for job scheduling of big data processing and we believe that it points out a new trend for large-scale data processing.**

***Index Terms*—Bandwidth-aware, big data, Hadoop, scheduling, software-defined networking (SDN).**

I. Introduction

SOFTWARE-DEFINED networking (SDN)[2] is a revolutionary network architecture that separates out network control functions from the underlying equipment and deploys them centrally on the controller, where OpenFlow is the standard interface. With SDN, applications can treat the network as a logical entity which makes enterprises and carriers gain unprecedented programmability, automation, and network control. In addition, SDN also provides a set of application programming interfaces (APIs) to simplify the implementation of common network services such as routing, multicast, security, access control, bandwidth management, quality-of-service (QoS), and storage optimization [3].

As a result, SDN creates amounts of opportunities to help enterprises build more deterministic, innovative, manageable, and high scalable data centers that extend beyond private enterprise networks to public IT resources, which makes implementing OpenFlow-based SDN data centers become a new developing trend nowadays.

At the same time, big data processing emerges as an important part of applications in such kind of data centers, where they not only handle but also generate amounts of data everyday. To concurrently process large-scale data with high efficiency, MapReduce with an open-source implementation named Hadoop[3] is proposed. It is an increasingly common computing system used by Yahoo!, Amazon, Facebook, etc. The logical view of Hadoop system is shown on the upper left of Fig. 1, while the physical view is shown at the bottom.

In practical Hadoop systems, one kind of issue that vitally impacts the overall performance is known as the NP-complete [4] minimum make span problem [5], [6], which exploits to find solutions on how to minimize the job completion time. Since network bandwidth is a scarce resource, assigning tasks on data local nodes is important for avoiding link occupation and then shortening the make span.

Many methodologies for enhancing data locality are proposed such as the Hadoop default scheduler (HDS) [1] and state-of-the-art balance-reduce scheduler (BAR) [2]. However, all of them either ignore allocating tasks in a global view or disregard available bandwidth as the basis for scheduling which results in losing optimized opportunities for task assignment.

As the new trend for big data processing evolves with SDN, one question to handle the minimum make span issue is naturally asked: Can we combine the bandwidth control capability of SDN with Hadoop system to exploit an optimized task scheduling solution that has high efficiency and agility in terms of job completion time for big data processing? (shown by the question mark on top of Fig. 1).

In this paper, we propose a bandwidth-aware task scheduler bandwidth-aware scheduling with SDN in Hadoop (BASS) to combine Hadoop with SDN. It first utilizes SDN to manage the bandwidth and allocates it in a time slot (TS) manner; then, BASS decides whether to assign a task locally or remotely

Manuscript received September 10, 2014; revised June 28, 2015; accepted October 24, 2015. Date of publication November 20, 2015; date of current version November 22, 2017. This work was supported in part by the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grant 2012BAH93F01, in part by the Innovation Research Fund of Huazhong University of Science and Technology under Grant 2014TS095, in part by the National Science Foundation of China under Grant 60803005 and Grant 91338201, and in part by the National High Technology Research and Development Program of China under Grant 2015AA015701.

P. Qin is with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China, and also with China Academy of Electronics and Information Technology, Beijing 100041, China (e-mail: qinpeng@hust.edu.cn).

B. Dai, B. Huang, and G. Xu are with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: daibin@hust.edu.cn; huangbx@hust.edu.cn).

Digital Object Identifier 10.1109/JSYST.2015.2496368

[1]Make span means the time between job's start time and job's finish time.

[2][Online]. Available: https://www.opennetworking.org/

[3][Online]. Available: http://lucene.apache.org/

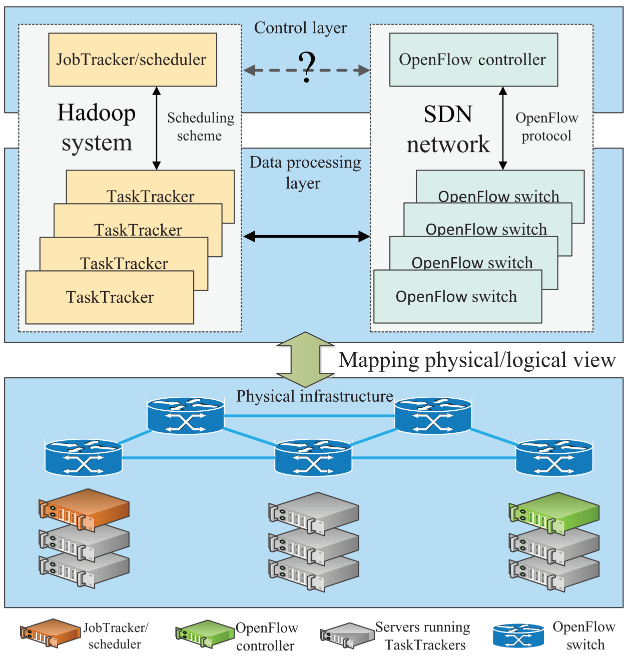

Fig. 1. Architecture of Hadoop big data processing with SDN.

depending on the completion time. It is not only able to guarantee data locality in a global view but also can efficiently assign tasks in an optimized way. The most important is that BASS takes a full consideration of the scarce network bandwidth from OpenFlow controller and regards it as a vital parameter for task scheduling. To our knowledge, BASS is the first to exploit talent of SDN for job scheduling of big data processing in Hadoop. Both examples and real-world experiments demonstrate that BASS outperforms all previous related algorithms including BAR which represents state of the art.

A. Contributions

The main contributions are summarized as follows.

1) We formalize the make span problem and develop a TS scheme for bandwidth allocation.
2) We exploit the capability of SDN and propose a bandwidth-aware task scheduler BASS which outperforms all previous related algorithms.
3) We provide *Example 1, Example 2, Example 3*, and implement extensive real-world experiments to demonstrate the effectiveness of BASS.

This paper is organized as follows. In Section II, we review some related work. In Section III, we formalize the problem of scheduling in Hadoop cluster. In Section IV, we propose the SDN-based bandwidth-aware scheduler BASS and present detailed examples for illustration. In Section V, we describe experimental details. Section VI concludes this paper and provides future expectations.

II. Related Work

A broad class of prior literatures ranging from big data processing to new emerging SDN is related to our work.

The Hadoop default scheduler searches for data local tasks greedily and assigns them to idle nodes [7] which, however, results in an increased job completion time. Matei *et al.* [8] propose delay scheduling to address the conflict between data locality and fairness. However, the introduced delays may lead to under-utilization and instability. Tan *et al.* [9] find that for current schedulers, map tasks and reduce tasks are not jointly optimized, which may cause job starvation and unfavorable data locality. To mitigate this problem, they propose a coupling scheduler to combine map and reduce tasks. Since Hadoop assumes that all cluster nodes are dedicated to a single user, it fails to guarantee high performance in shared environment. To address this issue, Seo *et al.* [10] proposed a prefetching and preshuffling scheme. However, the bandwidth occupation for transferring data block cannot be significantly reduced.

Jin *et al.* [2] proposed the BAR scheduler to globally reduce the job completion time which is most related to BASS. It is based on prior work [4] proposed by Fischer *et al.* to assign tasks efficiently in Hadoop. In [4], they investigate task assignment in Hadoop and give an idealized Hadoop model to evaluate the cost of task assignments. It is shown that task assignment is an NP-complete problem, which, however, can only be found a near-optimal solution with high computation complexity. To address this issue, BAR first produces an initial task allocation; then, the job completion time can be gradually reduced by tuning the initial task allocation. However, in some cases, such as *Discussion* shown in Section IV-B, BAR cannot efficiently reduce the job completion time, while BASS can reduce it from 39 to 35 s.

Independently, SDN originating from Clean Slate by University of Stanford is a new network architecture that separates out network control functions from the underlying equipment. The underlying switches perform only simple data forwarding. The leading SDN technology is based on the OpenFlow protocol [11], a standard that has been designed for SDN and already being deployed in a variety of networks and networking products [12], [13]. The most important feature of OpenFlow is its capability of network monitoring and traffic control which gives an alternative solution to speedup the Hadoop big data processing system.

Wang *et al.* [14] proposed application-aware networking focusing on configuring physical topology and routing for big data applications using SDN. Although it tries to explore the design of an SDN controller using a cross-layer approach that configures the network, job scheduling is out of its focus which is the biggest difference from BASS.

III. Problem Formalization

With the capability of network control provisioned by SDN, we can capture the real-time network status such as network traffic and bandwidth. We define some notations [15] shown in Table I.

TK_i denotes a task i within a Hadoop job; ND_j denotes a node j in the Hadoop cluster; SZ_i denotes the size of input split data for TK_i when it is assigned to ND_j; $TM_{i,j}$ denotes the data movement time of TK_i from data source $ND_{dataSrc}$ to ND_j; $TP_{i,j}$ denotes the time of task computation; $TE_{i,j}$

TABLE I
VARIABLES USED IN THE PAPER

Variables	Meanings
TK_i	A task i within a Hadoop job
ND_j	A node j in the Hadoop cluster
SZ_i	Size of input split data for TK_i assigned to ND_j
$TM_{i,j}$	Data movement time from data source $ND_{dataSrc}$ to ND_j
$TP_{i,j}$	Time of task computation
$TE_{i,j}$	Time of task execution
ΥI_j	The time when ND_j becomes idle
$\Upsilon C_{i,j}$	The completion time of TK_i
$BW_{j,k}$	Bandwidth between ND_j and ND_k
BW_{rl}	The real time available bandwidth of a link

denotes the time for task execution; ΥI_j denotes the time when ND_j becomes idle; $\Upsilon C_{i,j}$ denotes the completion time of TK_i; and $BW_{j,k}$ denotes the bandwidth between ND_j and ND_k, while BW_{rl} denotes the real-time available bandwidth of a link. Based on above symbols, we obtain

$$TM_{i,j} = SZ_i / BW_{dataSrc,j} \quad (1)$$

$$TE_{i,j} = TP_{i,j} + TM_{i,j} \quad (2)$$

$$\Upsilon C_{i,j} = TE_{i,j} + \Upsilon I_j. \quad (3)$$

For a map or reduce task TK_i, the objective function [shown in (4)] is to find an available node that can yield the earliest completion time among all n nodes of the cluster

$$ND_j = \text{argmin}_j \Upsilon C_{i,j} \quad (4)$$

where $1 \le j \le n$.

In the aspect of global view for a job, however, the objective function [shown in (5)] is a little different, where we need to find the slowest map or reduce task $TK_{i'}$ to minimize the completion time of a whole job

$$\min\{\Upsilon C_{i',j'} = \max \Upsilon C_{i,j} (1 \le i, i' \le m, 1 \le j, j' \le n)\} \quad (5)$$

where m is the task number of a job and n is the node number of the Hadoop cluster.

IV. SDN-BASED BANDWIDTH-AWARE SCHEDULING IN HADOOP FOR BIG DATA PROCESSING

A. TS Bandwidth Allocation

Benefiting from capability of SDN to obtain the real-time link bandwidth BW_{rl}, we propose a scheme to allocate bandwidth in a TS way. The main principle is described as follows.

Before Hadoop task scheduling begins, the occupation time of each link's residue bandwidth is disintegrated into equal TS, namely $TS_1, TS_2, \ldots, TS_k, \ldots$, duration of which is a tunable parameter according to practical network scenarios. We use SL_{rl} to denote the residue bandwidth at a certain TS for a certain link. If the task TK_i has the requirement of data movement through a certain path during (t_m, t_n), the scheduler will assign the corresponding TSs to it in advance guaranteeing that the bandwidths of all links on this path from starting slot TS_m ($t_m \in TS_m$) to ending slot TS_n ($t_n \in TS_n$) are reserved for TK_i.

The motivation for proposing TS scheme is as follows. Bandwidth is a scarce resource in practical Hadoop cluster especially when nodes compete drastically. Thus, to sufficiently utilize available bandwidth, we argue that always providing tasks requiring data movement with the most residue bandwidth and then taking it back after the occupation is a simple but effective solution in practice. Both *Example 1* and real-world experiments demonstrate its validity.

B. BASS: Bandwidth-Aware Scheduling With SDN in Hadoop

The BASS algorithm is illustrated as follows. A submitted job has m tasks and there are n available nodes in a Hadoop cluster. Note that the number of available nodes n may be less than the total nodes of the cluster especially when Hadoop system is shared by users.

Algorithm 1. BASS algorithm: **B**andwidth-**A**ware **S**cheduling with **S**DN in Hadoop

```
Require:
    Given the submitted job with $m$ tasks $TK_i$ and the $n$ nodes $ND_j$ in a Hadoop cluster.
Ensure:
1: for ($i = 1, 2, \ldots, m$) do
2:    Use capability of SDN to obtain the real-time bandwidth $BW_{rl}$ and the corresponding percentage of residue time slot $SL_{rl}$ for each link;
3:    Find $ND_{loc}$ with available idle time $\Upsilon I_{loc}$ for $TK_i$;
4:    Find $ND_{minnow}$ with available idle time $\Upsilon I_{minnow}$ for $TK_i$;
5:    if ($ND_{minnow} \equiv ND_{loc} \| \Upsilon I_{loc} \le \Upsilon I_{minnow}$) then
6:        Assign $TK_i$ to $ND_{minnow} \equiv ND_{loc}$;
7:    else if ($ND_{loc}$ is found $\&\& \Upsilon I_{loc} > \Upsilon I_{minnow}$) then
8:        Use Eq. (1) to (3) to calculate $\Upsilon C_{i,minnow}$ and the needed bandwidth $BW_{i,minnow}$ guaranteeing that $\Upsilon C_{i,minnow} < \Upsilon C_{i,loc}$;
9:        if ($BW_{i,minnow} \le BW_{rl}$) then
10:           Assign $TK_i$ to remote node $ND_{minnow}$;
11:           Assign $SL_{rl}$ of links on path from $ND_{dataSrc}$ to $ND_{minnow}$ and use Eq. (1) to calculate the slot number $TK_i$ needs;
12:       else
13:           Assign $TK_i$ to local node $ND_{loc}$;
14:       end if
15:   else if ($ND_{loc}$ is not found) then
16:       Assign $TK_i$ to remote node $ND_{minnow}$;
17:       Assign $SL_{rl}$ of links on path from $ND_{dataSrc}$ to $ND_{minnow}$ and use Eq. (1) to calculate the slot number $TK_i$ needs;
18:   end if
19: end for
20: return The assignment for all $m$ tasks.
```

Case 1: Data local node ND_{loc} is found

For a task TK_i, since bandwidth is a scarce resource in Handoop system, we prefer to assign it to a data local node ND_{loc} if there is one. When ND_{loc} is found, its available idle time ΥI_{loc} is recorded. However, ND_{loc} is not always the optimal option especially when there are already too many workloads on ND_{loc} which will let TK_i wait for a nontrivial extra time resulting in a much longer job completion time. In this scenario, we take the data movement time $TM_{i,j}$ into account and treat it as a significant parameter for job scheduling. Then, we search and find one node ND_{minnow} whose available idle time ΥI_{minnow} is minimum for the current time. ΥI_{minnow} is also recorded for further analysis.

Case 1.1: Data local node ND_{loc} is optimal

If data local node ND_{loc} is just the node ND_{minnow} or its available idle time ΥI_{loc} is no greater than ΥI_{minnow}, we assign TK_i to $ND_{loc} \equiv ND_{minnow}$ directly since there is no cost of data movement according to (1).

If data local node ND_{loc} is found but its available idle time ΥI_{loc} is greater than ΥI_{minnow}, there will be a tradeoff on whether to assign TK_i to node ND_{minnow} or not, depending on the residue bandwidth BW_{rl} (or SL_{rl} of TS) of links on path from data source $ND_{dataSrc}$ to ND_{minnow}. In this case to make sure that the remote node ND_{minnow} is a better choice than the data local node ND_{loc} for running task TK_i, we need to first calculate the task completion time ΥC_{minnow} and ΥC_{loc} using (1) to (3).

Subject to the objective that ΥC_{minnow} is smaller than ΥC_{loc} ($\Upsilon C_{minnow} < \Upsilon C_{loc}$), we obtain the corresponding bandwidth needed $BW_{i,minnow}$ for moving data to remote node ND_{minnow}. Then, we compare $BW_{i,minnow}$ with the real-time available bandwidth BW_{rl}.

Case 1.2: Remote node ND_{minnow} is optimal

If $BW_{i,minnow} \leq BW_{rl}$, it indicates that the available bandwidth is enough for transferring input data for TK_i with task completion time ΥC_{minnow} earlier than ΥC_{loc}. In this case, we assign TK_i to remote node ND_{minnow} and assign TSs SL_{rl} on path from data source $ND_{dataSrc}$ to ND_{minnow} according to SDN controller. Note that the TSs on a $link$ that are allocated to task TK_i are determined by the residue TSs of $path$ it belongs to, which are equal to the minimum residue TSs of all its links.

Case 1.3: $ND_{minnnow}$ is not optimal for limited bandwidth

If $BW_{i,minnow} > BW_{rl}$, it indicates that the available bandwidth in not enough for moving input data for TK_i with task completion time ΥC_{minnow} earlier than ΥC_{loc}. In this case, since the total cost of ND_{minnow} including the data transferring expense is much greater than data local node ND_{loc}, there is no need to run task TK_i remotely. Therefore, we assign TK_i to ND_{loc}.

Case 2: Data local node ND_j is not found (locality-starvation)

All above cases assume that ND_{loc} can be found and is available. However, Hadoop cluster may be shared by different users and each of whom is only authorized to use a subset of the whole nodes. Thus, the input split data have high probability of not being stored in any of these nodes. Therefore, ND_{loc} may not be found in this scenario which we call *locality-starvation* in this paper. To deal with it, *Algorithm 1* proposes a similarly solution like the case $BW_{i,minnow} \leq BW_{rl}$ where remote node ND_{minnow} is the optimal.

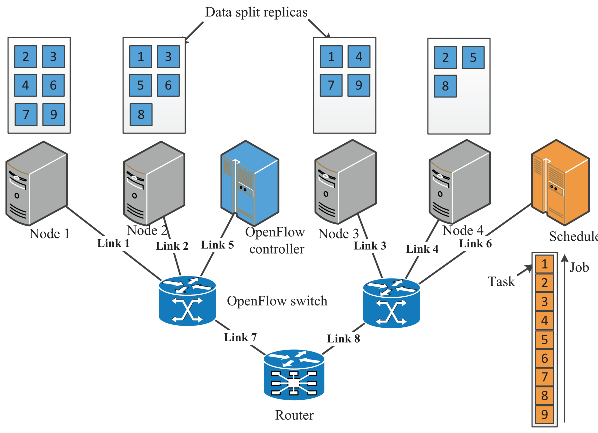

Fig. 2. Topology of a Hadoop cluster centrally controlled by SDN.

In this case, we assign TK_i to remote node ND_{minnow} and assign TSs SL_{rl} on path from data source $ND_{dataSrc}$ to ND_{minnow} according to SDN controller. The TSs on a $link$ that are allocated to task TK_i are also determined by the residue TSs of $path$ it belongs to, which are equal to the minimum residue TSs of all its links.

All the m tasks are scheduled via above process until an allocation result is obtained. BASS algorithm is an optimized scheduling scheme compared with the HDS and BAR schedulers for the following three reasons.

1) BASS maintains the priority of data locality with no cost of transferring data among nodes.
2) BASS utilizes SDN's bandwidth management capability to assign link bandwidth for a remote node ND_{minnow} making sure that the task completion time ΥC_{minnow} is earlier than that of a local node ND_{loc}.
3) BASS utilizes a TS scheme to allocate link bandwidth in a temporal dimensionality, which is a simple but effective solution in practice.

To explain this algorithm clearly, we give the following example.

Example 1: As is shown in Fig. 2, an OpenFlow controlled Hadoop cluster is composed of four task nodes, namely $Node1, Node2, Node3$, and $Node4$, an OpenFlow Controller and a Master Node/Scheduler. There are two OpenFlow switches, a router and eight links, namely $Link\ 1, Link\ 2, \ldots, Link\ 8$ connecting all nodes. A job with nine tasks $TK_i (i = 1, \ldots, 9)$ is scheduled by the master node. Each input split datum has two replicas located in two different nodes, respectively.

Assuming that size of each data block is 64 MB and each link bandwidth is 100 Mb/s, if the available bandwidth percentage SL_{rl} is 100%, then data movement time $TM_{i,j}$ calculated by (1) is 5.12 s. Here, we choose 5 s for simplification. We set each TS TS_k to be 1 s in this paper; thus, one data block movement occupies five TSs. Since each node is homogeneous, task computation time $TP_{i,j}$ is equal and we use 9 s for illustration in this example.

After scheduling starts, BASS allocates tasks $TK_i (i = 1, \ldots, 9)$ sequentially as Algorithm 1 describes and Fig. 3(a) shows the allocation result. Take the first task TK_1 for instance,

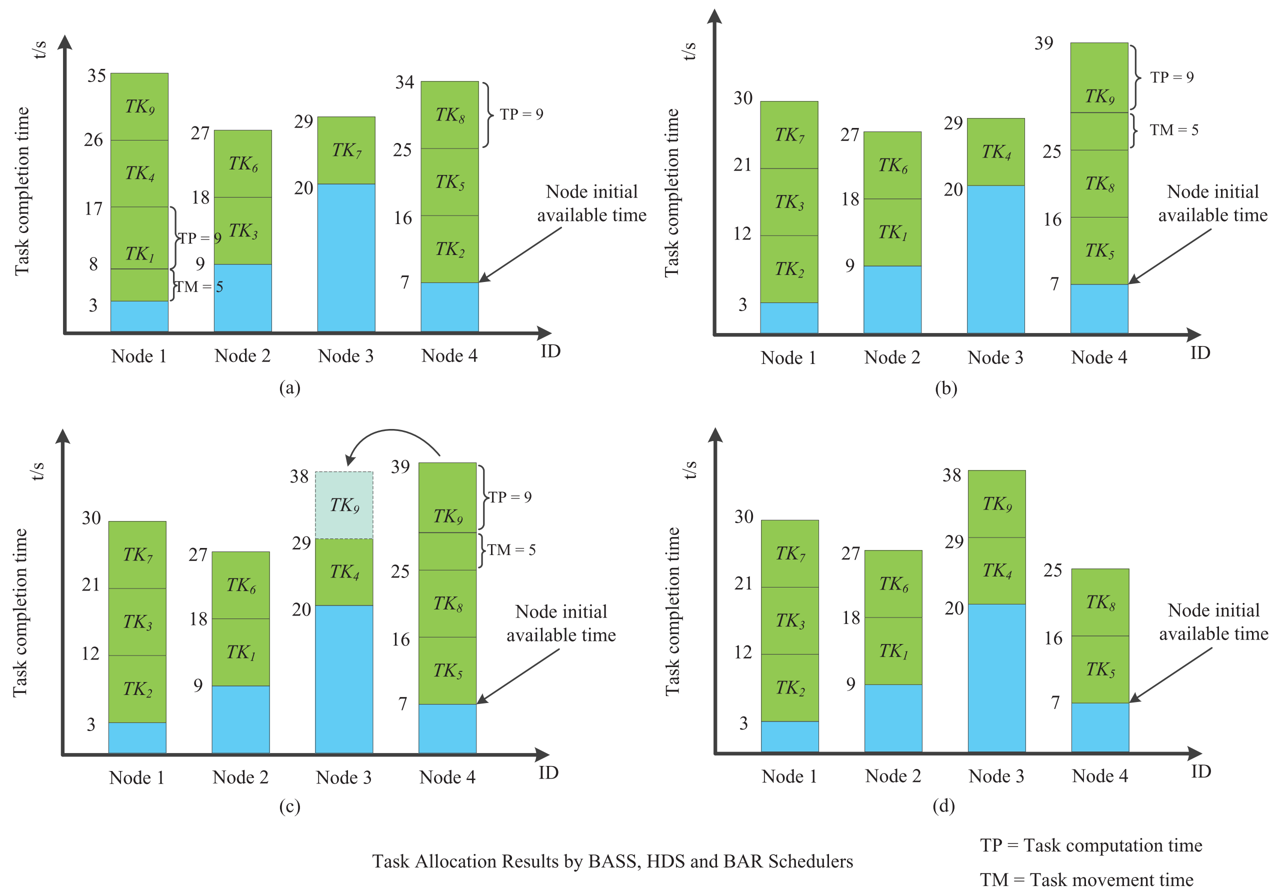

Fig. 3. (a) Task allocation result by BASS. (b) Task allocation result by HDS. (c) Loop 1 of task allocation by BAR. (d) Loop 2 and result of task allocation by BAR.

the available idle time (initial loads) of four servers are $\Upsilon I_{1,1} = 3$ s, $\Upsilon I_{1,2} = 9$ s, $\Upsilon I_{1,3} = 20$ s, and $\Upsilon I_{1,4} = 7$ s, respectively.

For TK_1, BASS first finds a node ND_2 with data locality and records its available idle time $\Upsilon I_{1,2} = 9$ s; then, it finds another node $ND_{minnow} = ND_1$ whose available idle time $\Upsilon I_1 = 3$ s is minimum for the current time. Since $\Upsilon I_2 > \Upsilon I_1$ which means that the available idle time of data local node ND_2 is greater than a remote node ND_1, it uses (1)–(3) to calculate the task completion time $\Upsilon C_{1,1}$ and $\Upsilon C_{1,2}$ running on the two nodes. By checking that the residue bandwidth of 100 Mb/s on Link 1 and Link 2 is enough for moving data block 1 with data transferring time $TM_{1,1} = 5$ s and $TM_{1,2} \approx 0$ s, it confirms that the task completion time $\Upsilon C_{1,1} = TM_{1,1} + TP_{1,1} + \Upsilon I_{1,1} = 5\,\text{s} + 9\,\text{s} + 3\,\text{s} = 17\,\text{s}$ running on remote node ND_1 is less than that of running on data local node ND_2 with $\Upsilon C_{1,2} = TM_{1,2} + TP_{1,2} + \Upsilon I_{1,2} = 0$ s $+ 9$ s $+ 9$ s $= 18$ s. Therefore, it allocates TK_1 to ND_1. BASS allocates other tasks in the same way. The allocation result is shown clearly in Fig. 3(a) where we can see the job completion time is 35 s since the last task TK_9 running on ND_1 determines the completion time of a whole job with $\Upsilon C_{9,1} = 35$ s.

In this case, BASS transfers input split data for TK_1 from ND_2; thus, the residue bandwidth on Link 1 that is 100% of 100 Mb/s from 3 to 8 s is allocated for data movement. This means that the occupied TSs consist of TS_4, TS_5, TS_6, TS_7, and TS_8. The occupation of TSs on Link 2 is the same.

Note that we may also choose ND_3 to transfer input split data for TK_1. In this case, Link 1, Link 7, Link 8, and Link 3 need to allocate TSs for data movement where the occupation is also the same as before.

Discussion 1: To see the efficiency of BASS scheduler we choose HDS and BAR scheduler that stands for state of the art in this domain to assign the same nine tasks for comparison. Also assuming that data movement time is 5 s and task computation time is 9 s.

HDS always chooses a task TK_i for ND_j with data locality and assigns TK_i to it. If no data local task is available, HDS scheduler will choose a task randomly for ND_j. Take TK_1, for instance, replicas of its input split data are stored at ND_2 and ND_3, after comparing the available idle time of them, HDS knows $\Upsilon I_2 = 9\,\text{s} < \Upsilon I_3 = 20\,\text{s}$; thus, it allocates TK_1 to node ND_2. Similarly, remaining tasks are assigned. Note that, when ND_4 is available at 25 s, only nonlocal TK_9 is left, then ND_4 has to carry it out. Finally, ND_1 executes TK_2, TK_3, TK_7; ND_2 executes TK_1, TK_6; ND_3 executes TK_4; ND_4 executes TK_5, TK_8, TK_9. Since TK_9 running on ND_4 determines the completion time of the whole job, we see that the job completion time is 39 s which is 4 s later than that of BASS scheduler [shown in Fig. 3(b)].

BAR scheduler is based on HDS and it further improves job performance by globally adjusting data locality according to network state and cluster workload. In the first phase, BAR

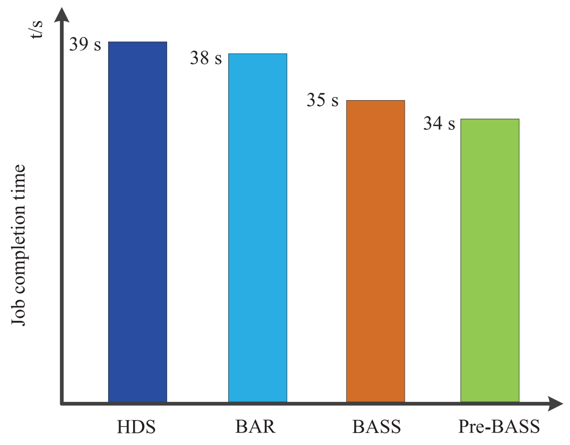

Fig. 4. Performance comparison of schedulers: HDS, BAR, BASS, and pre-BASS.

allocates tasks $TK_i (i = 1, \ldots, 9)$ to nodes $ND_j (j = 1, \ldots, 4)$ obeying data locality principle with the same result shown in Fig. 3(b). In the second phase, BAR searches for the task TK_{lat} whose completion time $\Upsilon C_{i,lat}$ is the latest and checks if there is a remote node ND_{remo} with the completion time $\Upsilon C_{i,remo}$ earlier than $\Upsilon C_{i,lat}$. If ND_{remo} is found, BAR assigns TK_i to it and repeats this process until no such node is available, using the same parameters aforementioned and taking the second phase of BAR scheduler for instance. Since the data local assignment is obtained in the first phase, BAR knows that TK_9 on ND_4 is the latest one with completion time $\Upsilon C_{9,4} =$ 39 s [shown in Fig. 3(c)]. It then checks if ND_3 with available idle time $\Upsilon I_3 = 25$ s can run task TK_9 with completion time $\Upsilon C_{9,3} < 39$ s? Fortunately, $\Upsilon C_{9,3} = TM_{9,3} + TP_{9,3} + \Upsilon I_3 = 0\,\text{s} + 9\,\text{s} + 29\,\text{s} = 38\,\text{s}$ is smaller than 39 s. Therefore, BAR moves TK_9 from ND_4 to ND_3 with job completion time being 38 s, as is shown in Fig. 3(d).

From this case, we can see clearly that BASS scheduler outperforms BAR and HDS to improve the overall performance in terms of job completion time. A comparison of HDS, BAR, and BASS is summarized in Fig. 4. To further demonstrate this, extensive experiments for jobs with different task size are carried out in Section V.

Discussion 2: Can we further reduce the job completion time?

Seo *et al.* [10] proposed a prefetching scheme to improve the overall performance under shared environment while retaining compatibility with the native Hadoop scheduler. Inspired by this idea, we propose to use a similar prefetching method called pre-BASS to further reduce the job completion time.

The main process is described as follows. First, the optimized pre-BASS scheduler allocates tasks guaranteeing that each one is optimal in terms of its completion time. Then, pre-BASS checks each data-remote task TK_{remo} and let its input split data prefetched/transferred before the available idle time ΥI_{remo}, as early as possible which depends on the real-time residue bandwidth BW_{rl} or SL_{rl}. Note that when prefetching a data block, it is always moved from the least loaded node storing the replica to minimize the impact on the overall performance. However, we should note that pre-BASS is not always better than BASS since the former let task node store the input split data for longer time which may introduce negative influence to the TS scheme for bandwidth allocation. We give another example to illustrate pre-BASS scheme.

Example 2: Thinking about the first task TK_1 of BASS running on remote node ND_1 [as Fig. 3(a) shows] in *Example 1*. The data movement starts at 3 s and it occupies five TSs, namely $TS_4, TS_5, TS_6, TS_7, TS_8$. If we utilize the prefetching scheme here to let this task prefetch its input data at 0 s and occupy TSs $TS_1, TS_2, TS_3, TS_4, TS_5$, then the completion time of all tasks on this node will be reduced from 35 to 32 s. In this way, the last finished task will not be TK_9 but TK_8, and the job completion time will not be 35 s but 34 s which is a further performance improvement in the global view (see the right side of Fig. 4 for performance comparison).

Discussion 3: How can we make the utmost of SDN?

One important feature of SDN/OpenFlow is its simple QoS scheme via a queuing mechanism. Recall the huge volume of shuffling traffic which consumes large amounts of bandwidth. If Hadoop traffic especially the shuffling traffic is provided with higher priority utilizing the QoS capability of OpenFlow [16], we believe that the overall performance of Hadoop system in terms of job completion time will be further reduced. We also give an example to illustrate it.

Example 3: Take the topology of SDN-controlled Hadoop cluster in Fig. 2 for instance. We first set the maximum rate of both OpenFlow switches to be 150 Mb/s and setup three queues: Q_1 with 100 Mb/s, Q_2 with 40 Mb/s, and Q_3 with 10 Mb/s. Then, new flow entries are added to direct shuffling traffic to Q_1 that has higher link bandwidth and to direct background traffic to Q_3 to limit its impact on Hadoop task. The rest of traffic occupy Q_2. This simple scheme outperforms that of putting all traffic in the same queue with maximum rate of 150 Mb/s which is the default scheme.

V. Experiments for Performance Evaluation

In this section, we present real-world experiments to investigate the effectiveness of BASS. For comparison, two prior most-related schedulers HDS and BAR described in Section IV are also implemented.

A. Experimental Setup

In this experiment, the Hadoop cluster with OpenFlow switches is shown in Fig. 2. The cluster consisting of six nodes located in five physical systems runs the Hadoop version 1.2.1 connected to 2 Open vSwitch (OVS).[4]

The number of block replicas is set to be 3. The size of data block is 64 MB. The maximum link rate is set to be 100 Mb/s which is a tunable parameter in practice.

We utilize the *ProgressRate* scheme [17] which works well in practice to estimate the initial workload and the available idle time ΥI of each node. The *progress rate* of each task is calculated by $ProgressRate = ProgressScore/T$, where *ProgressScore* represents the task progress between 0 and 1; T

[4][Online]. Available: http://openvswitch.org/

TABLE II
COMPARISON OF DATA LOCALITY AND JOB COMPLETION TIME AMONG BASS, BAR, AND HDS SHEDULERS

Data size	BASS				BAR				HDS			
	MT (s)	RT (s)	JT (s)	LR (%)	MT (s)	RT (s)	JT (s)	LR (%)	MT (s)	RT (s)	JT (s)	LR (%)
(a) Wordcount jobs												
150M	58	52	78	33.3	59	47	78	33.3	65	53	78	33.3
300M	79	76	128	66.7	83	105	146	66.7	89	107	156	50
600M	149	192	231	58.3	183	220	259	66.7	193	232	269	83.3
1G	275	216	298	76.2	290	230	305	76.2	293	232	311	57.1
5G	1190	1164	1302	75.2	1320	1218	1377	75.2	1347	1252	1396	77.2
(b) Sort jobs												
150M	24	43	55	50	25	49	67	25	28	62	74	50
300M	26	65	91	66.7	47	98	110	66.7	54	98	117	50
600M	79	123	144	50	90	135	155	50	100	148	168	50
1G	147	254	262	56.3	150	261	285	56.3	152	297	323	62.5
5G	640	1531	1572	66.3	660	1600	1632	71	772	1730	1859	63.7

MT, map phase completion time (s); RT, reduce phase completion time (s), JT, job completion time (s), data locality ratio (LR) $= \frac{\text{data local task number}}{\text{total task number}}$.

is the amount of time the task has been running for. The time to complete is then estimated by $\Upsilon I = \frac{(1-ProgressScore)}{ProgressRate}$.

We choose both Wordcount and Sort jobs[5] as our test case and run them for different workload with data size to be 150 MB, 300 MB, 600 MB, 1 GB, and 5 GB, respectively. We repetitively execute a background job to provide each test with initial workload. Each test is run for 20 times and we use the average value for comparison.

B. Experimental Results

From Table II, we see that for Wordcount job, BASS scheduler always finishes with the minimum map and reduce phase completion time, namely the minimum make span compared with BAR and HDS. In some cases, data locality ratio (LR) of BASS is low, taking the input data of 600 M as an example, LR of BASS is 58.3%, while LRs of BAR and HDS are 66.7% and 83.3%. However, the make span of BASS is only 231 s compared with 259 s of BAR and 269 s of HDS.

The reason is that in this scenario, bandwidth resource is sufficient for transferring data and computation resource on data local node ND_{loc} is scarce, thus running TK_i on ND_{loc} is not a good choice anymore. In this sight, we argue that link bandwidth and data locality should be taken integratedly into account to achieve the best overall performance in practice.

Sort job in Table II also demonstrates the validity of BASS scheduler in real-world system.

A clear comparison of above three schedulers is shown in Fig. 5 for both Wordcount and Sort jobs, where we can see that BASS outperforms the other two in terms of job completion time.

To see how the bandwidth will affect job completion time, more experiments are carried out. We try to adjust bandwidth of each link to see the influence on performance of BASS. However, there is no significant difference on Job completion time. It is mainly because the link condition is always relatively satisfactory in each case using the TS scheme.

[5]We choose wordcount and sort for test because the former consumes more CPU while the later occupies more disk I/O resources.

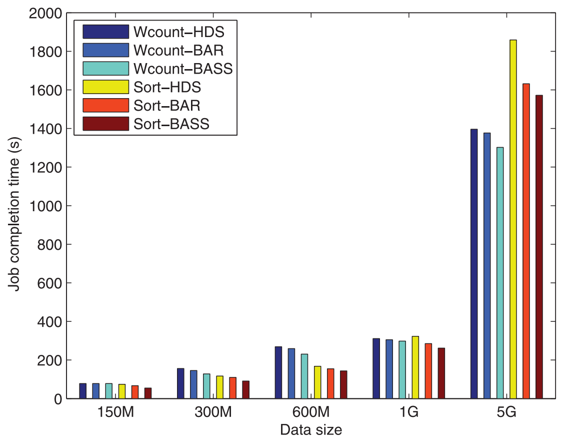

Fig. 5. Job completion time for both wordcount and sort jobs.

VI. CONCLUSION AND EXPECTATIONS

In this paper, we exploit to utilize SDN and take a full account of link bandwidth for improving performance of big data processing. We first formalize the make span problem in Hadoop and develop a TS scheme for bandwidth allocation. Then, we propose the SDN-based bandwidth-aware scheduler BASS which can flexibly assign tasks in an optimized way. Last but not least, we provide examples and implement extensive real-world experiments to demonstrate the effectiveness of BASS. To our knowledge, BASS is the first to exploit talent of SDN for job scheduling of big data processing in Hadoop cluster.

As the evolvement of SDN with big data processing, for future work we plan to implement BASS in enterprise's data centers composed of practical SDN products such as the OpenFlow switch and we will evaluate BASS's scalability in a much larger network cluster. Furthermore, we believe that BASS points out a new trend for large-scale data processing.

REFERENCES

[1] T. White, *Hadoop: The Definitive Guide*, 3rd ed. O'Reilly Media Inc., May 2012.
[2] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "Bar: An efficient data locality driven task scheduling algorithm for cloud computing," in *Proc. 11th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput. (CCGrid'11)*, Newport Beach, CA, USA, May 2011, pp. 295–304.

[3] S. Azodolmolky, P. Wieder, and R. Yahyapour, “Cloud computing networking: Challenges and opportunities for innovations,” *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 54–62, Jul. 2013.
[4] M. J. Fischer, X. Su, and Y. Yin, “Assigning tasks for efficiency in hadoop: Extended abstract,” in *Proc. 22nd Annu. ACM Symp. Parallel. Algorithms Architect. (SPAA’10)*, Thira, Santorini, Greece, Jun. 2010, pp. 30–39.
[5] K. Birman, G. Chockler, and R. van Renesse, “Toward a cloud computing research agenda,” *SIGACT News*, vol. 40, no. 2, pp. 68–80, Jun. 2009.
[6] E. Bortnikov, “Open-source grid technologies for web-scale computing,” *SIGACT News*, vol. 40, no. 2, pp. 87–93, Jun. 2009.
[7] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Job scheduling for multi-user mapreduce clusters,” EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-55, Apr. 2009 [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-55.html
[8] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling,” in *Proc. 5th Eur. Conf. Comput. Syst. (EuroSys’10)*, Paris, France, Apr. 2010, pp. 265–278.
[9] J. Tan, X. Meng, and L. Zhang, “Coupling task progress for MapReduce resource-aware scheduling,” in *Proc. IEEE INFOCOM*, Turin, Italy, Apr. 2013, pp. 1618–1626.
[10] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng, “HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment,” in *Proc. IEEE Int. Conf. Cluster Comput. Workshops (CLUSTER’09)*, New Orleans, LA, USA, Sep. 2009, pp. 1–8.
[11] N. Mckeown *et al.*, “Openflow: Enabling innovation in campus networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Jan. 2008.
[12] M. Olson. (2010, May). *Hadoop*: *Scalable Flexible Data Storage and Analysis* [Online]. http://www.cloudera.com/content/cloudera/en/resources/library/whitepaper/hadoop_scalable_flexible_storage_and_analysis.html
[13] S. Narayan *et al.*, “Openflow enabled hadoop over local and wide area clusters,” in *Proc. SC Companion: High Perform. Comput. Netw. Storage Anal. (SCC)*, Salt Lake City, UT, USA, Nov. 2012, pp. 1625–1628.
[14] G. Wang, T. S. E. Ng, and A. Shaikh, “Programming your network at runtime for big data applications,” in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN’12)*, Aug. 2012, pp. 103–108.
[15] Z. Guo and G. Fox, “Improving MapReduce performance in heterogeneous network environments and resource utilization,” in *Proc. 12th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput. (CCGrid)*, Ottawa, ON, Canada, May 2012, pp. 714–716.
[16] S. Narayan, S. Bailey, and A. Daga, “Hadoop acceleration in an openflow-based cluster,” in *Proc. SC Companion: High Perform. Comput. Netw. Storage Anal. (SCC)*, Salt Lake City, UT, USA, Nov. 2012, pp. 535–538.
[17] The Apache Software Foundation. (2014). *Hadoop* [Online]. Available: http://lucene.apache.org/hadoop

Peng Qin, photograph and biography not available at the time of publication.

Bin Dai, photograph and biography not available at the time of publication.

Benxiong Huang, photograph and biography not available at the time of publication.

Guan Xu, photograph and biography not available at the time of publication.