Network-Aware Resource Management for Scalable Data Analytics Frameworks

Thomas Renner, Lauritz Thamsen, Odej Kao
Technische Universität Berlin, Germany
{firstname.lastname}@tu-berlin.de

Abstract—**Sharing cluster resources between multiple frameworks, applications and datasets is important for organizations doing large scale data analytics. It improves cluster utilization, avoids standalone clusters running only a single framework and allows data scientists to choose the best framework for each analysis task. Current systems for cluster resource management like YARN or Mesos achieve resource sharing using containers. Frameworks execute their tasks in containers, which they receive after specifying resource requirements. Currently, the container placement is based on available computing capabilities in terms of cores and memory, yet neglects to also take the network topology and data locations into account.**

In this paper, we propose a container placement approach that (a) takes the network topology into account to prevent network congestions in the core network and (b) places containers close to input data to improve data locality and reduce remote disk reads in a distributed file system. The main advantages of introducing topology- and data-awareness on the level of container placement is that multiple application frameworks can benefit from improvements. We present a prototype integrated with Hadoop and an evaluation with workloads consisting of different applications and datasets using Apache Flink. Our evaluation on an 64 core cluster, in which nodes are connected through a fat tree topology, shows promising results with speedups of up to 67% for workloads, in which the network is an important resource.

I. Introduction

More and larger datasets become available through data-sensing mobile devices, wireless sensor networks, software logs, and user-generated content. Gaining insights into the data is becoming increasingly relevant for more and more applications. For analyzing this data quickly and efficiently, different scalable data analytic frameworks have been developed. Prominent examples include MapReduce [1], Dryad [2], Spark [3], Flink [4], Naiad [5], and Storm [6]. Each of these frameworks has advantages and disadvantages, and the best choice depends on the type of data analysis task (e.g. batch, stream, or graph processing), the structure and size of the data, as well as the data scientists skills. Thus, there is a growing need to share cluster resources between different users and frameworks. For this reason, the frameworks typically run within containers on cluster resource management systems like YARN [7] or Mesos [8], where they grant rights to use specific amounts of resources. Furthermore, the data that is to be analyzed typically resides in a co-located distributed file system such as HDFS [9]. This design is attractive for companies and data center providers, because it allows to run workloads consisting of different applications and multiple frameworks on the same datasets in a shared cluster. It also provides higher resource utilization [10] and enables multi-tenancy. Thus, different users and organizations can share a cluster in a cost-effective manner [11]. Furthermore, from the user's perspective, data scientists also have more freedom to choose the most appropriate of the different frameworks for their analysis task at hand.

When the cluster size increases and tasks are highly distributed in containers, it is likely that data needs to be accessed from remote disks and tasks communiate with each other over the network. This can cause a lot of network traffic, especially for data-intensive applications. Various authors identified the network as a major bottleneck for distributed data analytics. Some introduced task schedulers that preserve data locality by placing tasks on nodes close to their input data [11]–[16]. Others introduced network optimizations including load balancing in multi-path networks or network traffic prioritization [17]–[20]. However, most of these works are implemented on the framework level instead of the level of resource management and, thus, support only standalone clusters running a single framework. Moreover, to our best knowledge none of these works explicitly addresses container placement in hierarchical networks on the resource management level.

Large clusters for data analytics are typically organized in hierarchical network topologies like fat trees [21]. In such designs, nodes are grouped into racks of 20-80 nodes at the lowest level and multiple paths with different hop counts can exist between two nodes [14]. Typically, bandwidth between nodes within a rack is higher than bandwidth between nodes in different racks, because paths between switches are shared by more nodes [11]. Despite, container placement in many resource management systems is based on compute resource profiles without taking information on the network topology or data locality into account, e.g. requests that require 10 containers with 4096 MB memory and 2 virtual cores. However, even with optimal compute scheduling the network can become a bottleneck [22]. Especially, when multiple data-intensive applications run in parallel on a cluster.

With this motivation, we explore the problem of sharing resources between multiple workloads in hierarchical networks. In particular, we present a network-aware container placement approach for data analytics applications. A key advantage of improving container placement is that workloads consisting of different frameworks, applications and datasets in a shared cluster can benefit from this optimization.

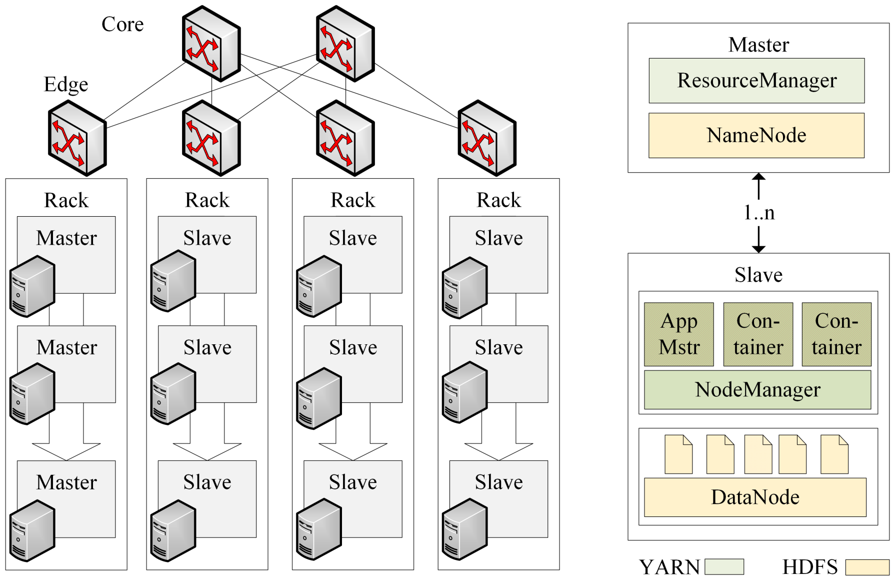

Fig. 1: Data analytic architecture based on Apache Hadoop [23] and organized in a fat tree topology [21]

We use simmulated annealing [24] to find acceptably good container placements based on a weighted cost function in a fixed amount of time. The cost function tries to optimize network by improving network bisection bandwidth and relies on two main factors:

1) Data Locality: placing containers near to the job input data, so frameworks can place source tasks locally to input data.
2) Container Closeness: placing containers close to each other with a few as possible network links between different racks involved.

We weighted the factors of this cost function for workloads in which the network is an important resource. This is, for example, often the case with iterative programs [5] [25] as used in machine learning and graph analysis. We implemented our approach on top of YARN and evaluated it with workloads consisting of Flink jobs, which allows us to run iterative programs. The experienced performance improvements for such workloads lie between 39 - 67 %.

The rest of the paper is organized as follows. Section II introduces the background and motivation of our network-aware container placement approach. The system architecture is presented in Section III. Section IV presents the design of our scheduler in more detail. Evaluation and results are given in Section V. Related work is discussed in Section VI. Section VII concludes this paper with remarks on future work.

II. Background and Motivation

This section describes a typical cluster setup for scalable data analytics as the targeted environment for our container placement approach. First, we have a closer look at the physical infrastructure focusing on network aspects. Then, we discuss the typical software stack and architecture as well as resulting challenges based on Apache Hadoop [23], on which our proposed architecture, prototype implementation and evaluation is based.

Large clusters with hundreds of nodes are often organized in hierarchical multi-path networks [14]. One widely used type of topology for such networks is the fat tree topology [21]. In such a topology, also shown in Figure 1, physical nodes are grouped into racks of 20 - 80 nodes at the lowest level by commodity switches. Furthermore, multiple paths and different hop counts between two nodes can exist. In addition, bandwidth capacities become higher as one moves up the tree. However in data analytic clusters, the available bandwidth between nodes in the same rack can be higher than the bandwidth between nodes located in different racks. The reason is that links between racks are shared by more nodes at the same time, which can result in network congestion on the core layer links [11]. Therefore, it is preferable to place applications close to each other as well as close to input data to avoid network traffic on the upper layers as much as possible.

Two main components of Hadoop [23] are the cluster resource management system YARN [7] and the distributed file system HDFS [9]. Typically, both system are running co-located, allowing mixed workloads consisting of different frameworks, applications and datasets on a shared cluster. Both systems follow the master/slave pattern. The master node oversees and coordinates data storage and computing functionalities. The slave nodes make up the majority of hosts and do the work of storing the data and running computations within containers. In the following, we will discuss both systems as shown in Figure 1 in more detail and highlight the motivation for our container placement approach.

YARN consists of four main components: ResourceManager, NodeManager, Application Master and Container. The ResourceManager is YARN's master component. It is the central arbitrator of all resources and is responsible for scheduling application workloads on available resources. The scheduling function is based on an application resource request specifying the amount of needed containers as well as memory and CPU requirements per container. Currently, the scheduler takes no network information into account and, thus, treats the network as a black box. A NodeManager running per-node is responsible for monitoring containers and their memory and CPU usage, reporting to the Resource Manager. Together, the central Resource Manager and the collection of Node Managers create the computational infrastructure. An ApplicationMaster running per-application has the responsibility of requesting and receiving appropriate resource containers from the Resource Manager. The negotiation and the placement does not take network or data locality into account. This can lead to high container distributions over the network and, thus, more network traffic, because of remote reads from a distributed file system and communication among tasks on different nodes. In addition, the ApplicationMaster tracks the application status and monitors its progress (e.g. if an application is finished, cancelled or still running). A Container represents successfully allocated resources, provided by the Resource Manager. A container grants rights to an application to use a specific amount of resources on a node, and, thus, provides an execution environment. Typically, one framework worker is executed in one container.

The cluster resource management system is typically co-located with the distributed file system HDFS [9]. In such file systems, data is replicated over multiple nodes for fault-tolerance and faster access. Many frameworks try to exploit data locality by scheduling tasks close to the input data to maximize system throughput, because available network bandwidth is often lower than the clusters disk bandwidth [26]. HDFS consists of two main components, namely NameNode and DataNode. A single NameNode acts as a master that regulates access to files by clients and manages the file system name space. Multiple DataNodes, usually one per node, manage storage attached to the nodes that they run on. A file is split into one or more blocks and these blocks are stored replicated on DataNodes.

Data analytic frameworks such as MapReduce [1], Spark [3], or Flink [4], or Naiad [5] are typically executed on top of a cluster resource management system and a distributed file system. In general, these frameworks split data across many nodes, process them data-parallely and finally transfer and merge intermediate results. Thus, they allow to process a huge amounts of data, but also generate a lot of network traffic.

III. System Architecture

This section describes the system architecture of our container placement approach. The architecture is based on YARN and HDFS, which are booth introduced in Section II.

As shown in Figure 2, the architecture follows a master and slave model. The slave nodes make up the majority of nodes and do the work of storing the data and running the computations within container. Therefore, each slave node hosts a NodeManager and DataNode daemon. The master nodes consist of a ResourceManager, NameNode and a SDN network controller [27]. They manage data storage as well as network and computing functionalities. The ResourceManager receives job submissions from the users, and is responsible for allocating needed resources and containers. The SDN network controller automatically gets the current network topology and could be used for flow switching in future work. The NameNode provides information about block localities, which is used to improve data locality. The Placement component is the focus of this paper and contains the logic for our placement algorithm. It decides where to place an application and its container best, based on available resources, topology information, block location and running applications. The component uses existing REST interface to all master components. The logic of our placement algorithms is described in Section IV in more detail

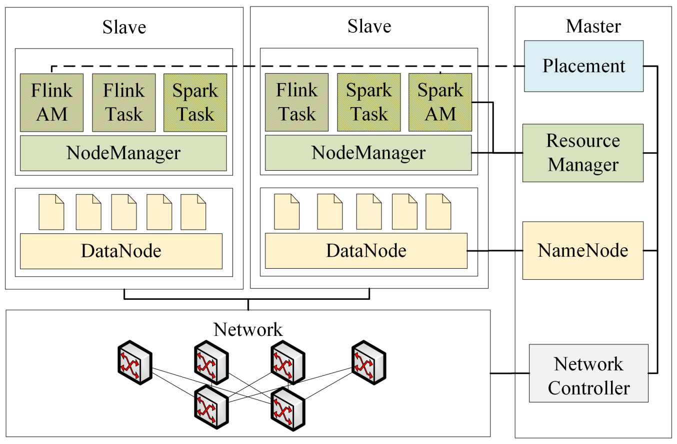

Fig. 2: Architecture Overview

Figure 3 shows an sequence diagram to illustrate the flow between all system components to find a good container placement. First, an application is submitted to YARN's ResourceManager. The submission contains the amount of needed containers and their computation specification as well as the path of the input data. The ResourceManager deploys an ApplicationMaster on the available slave nodes. The AM is responsible for allocating containers from the ResourceManager. Therefore, it first asks our placement component where to place containers. The placement component receives an application request specified with a resource profile, which contains the amount of needed containers and its virtual cores and memory demand per container as well as distributed file input path. Afterwards, the placement component calculates application container (i. e. placement hints) based on information provided by the ResourceManager, Network Controller and NameNode including available resources, running application containers, distributed file system path and network topology. The result are placement hints that are send back to Application Master, which afterwards sends container requests for these hints to the ResourceManager. Finally, the ResourceManager will allocate the containers and the execution of the application can begin.

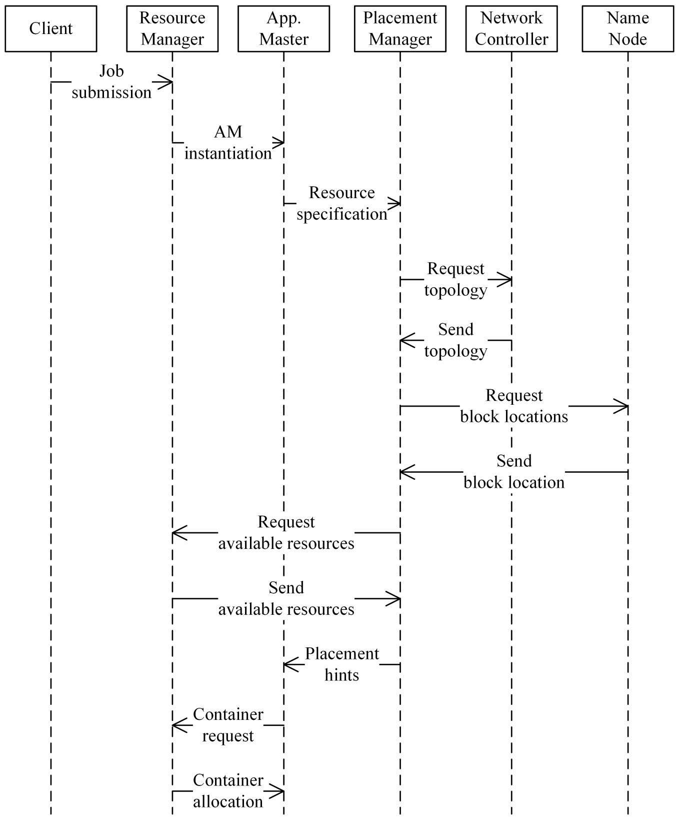

Fig. 3: Sequence diagramm

IV. Network-aware Container Placement

The goal of our network-aware container placement is to improve the execution time of workloads in clusters shared by multiple data analytics frameworks. This is achieved by incorporating the following two objectives into the placement:

1) Container togetherness: Placing containers of applications close together with a minimal number of network hops in between them to reduce network traffic on links in the core network.
2) Data locality: Placing containers close to their application's input data to allow more local reads from the underlying distributed file system as remote reads cause network traffic.

Finding good placements for containers in large clusters is an optimization problem with a potentially huge search space [28]. For this reason, we use simulated annealing [24], a probabilistic method to find an approximation of the global optimum of the given function in a fixed amount of time. The cost function we propose consists of two main components, reflecting the two objectives of network-aware placement we identified: container togetherness and data locality. A third minor component is derived from the fact that containers share the resources of the nodes they are placed on. Consequently, the container load should be balanced over available resources. All three components are normalized and assigned weights, depending on their importance and the cluster infrastructure.

A. *Placing Containers Close Together*

Different container placements result in different communication paths in hierarchical networks. For instance, placing an application and its containers on a single rack involves only the single top-of-rack switch, so all traffic is kept on rack level and network congestions on the core network can be avoided. Reducing the communication over the network links on the core layer is critical because the available bandwidth between nodes within a rack is typically significantly higher than the bandwidth between nodes across racks [11]. Thus, it is preferable to place an application and its containers close to each other with a small number of links on the core network involved. In the best case, a single rack with sufficient resources is available for the application request. If this is not the case, our approach places containers over multiple racks but minimizes the necessary hops on the core level.

For the first component of the cost function we sum up the involved hops on the core layer of the network. Therefore, we determine the shortest paths between all container pairs and sum up all shortest paths greater than two hops, thereby excluding paths between containers connected to the same switch. This excludes the links between hosts and switches to focus the cost function on links in the core layer, in which traffic aggregates much more significantly as more nodes share these links.

Figure 4 shows an example in which eight nodes are connected using six switches that form a hierarchical multi-path network. Each node can host up to four containers. A placement of twelve containers results in eight containers on Rack 1 and four in Rack 2. The network cost–being the sum of the shortest paths among all container pairs–in this example is given by 32 hops for this placement.

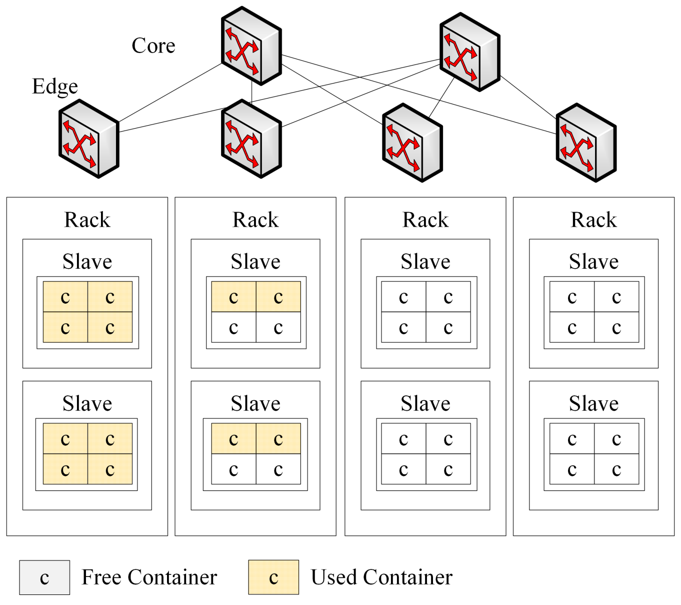

Fig. 4: A container placement with low network costs.

B. Placing Containers Close to Inputs

Most frameworks try to archieve data locality by placing source tasks on top of input data when possible. This can reduce network traffic and improve job execution time. Reading files or blocks locally is only constrained by the disk read speed, not additionally by the network throughput. Therefore, our goal is to place containers in a way that we can allow data locality for the applications running within the containers. Optimally, each container placement covers all of the blocks of the application's input data. Furthermore, as blocks are usually replicated to provide fault-tolerance, placements can cover multiple replicas per block. Covering more than one replica introduces degrees-of-freedom for the framework's scheduling that often has to satisfy other constraints. For this reason, our cost function considers both how many blocks and how many block replicas are covered by a placement. Covering all blocks is more important than covering many replicas. Therefore, the ratio of blocks covered is given more weight than the amount of replicas covered in this cost function component.

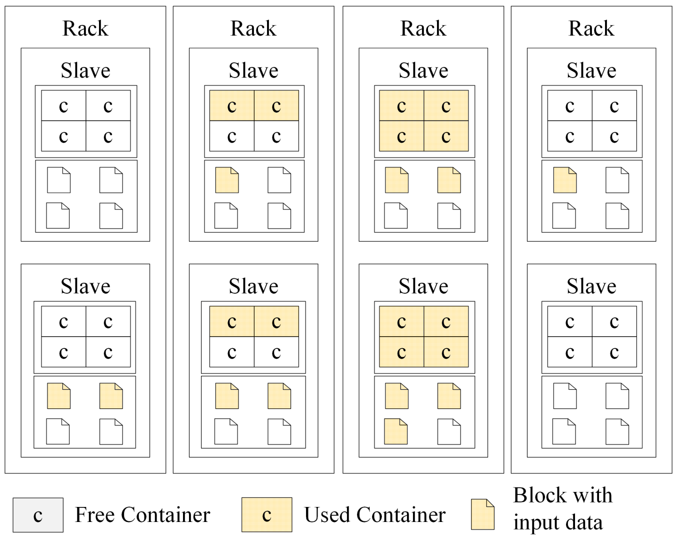

Fig. 5: A container placement with high data locality.

Figure 5 shows an example of a cluster consisting of eight nodes. Each node can host up to four containers and store up to four distributed file system blocks. The containers of the application running in the cluster are placed with considerable fractions of their input locally available.

C. Placing Containers Close To Inputs, Close Together and Balanced Over Available Resources

The containers provided by resource management systems typically only provide weak isolation between containers. Subsequentely, tasks executed in containers compete for the resources of nodes. These resources include the computing capabilites of nodes, particularly the cores and memory, as well the I/O capabilites, particularly disk and network I/O. This requires to distribute containers evenly over available resources.

At the same time, different tasks utilize the different resources to different amounts. Sorting operations, for example, often require lots of memory whereas map tasks can, for instance, be mainly CPU-bound. Since systems for large scale data analysis are often based on data parallelism [1], [2], [29], tasks of the same application tend to stretch the same resources at the same time. For this reason, resources are ideally shared by containers of different applications. In other words, containers of a single application should be placed on as many nodes as possible. Optimizing this third objective is sometimes naturally in conflict with optimizing the two main components of our cost function. Yet, as all three components of the cost function are weighted and the third component getting only little weight, it only functions as a tipping point towards balanced solutions among placements with roughly equal network and data input costs.

Assigning sensible weights to the two main components of our cost functions depends on the cluster configuration. In particular, appropriate weights depend on the disk read speed of the nodes and the throughput of network links in the cluster network.

Figure 6 shows an example in which eight nodes are connected using six switches that form a hierarchical multi-path network. Each node can host up to four containers and store up to four distributed file system blocks. The containers of the application running in the cluster are placed such that network costs are minimal, data locality is high and the involved resources are utilized evenly.

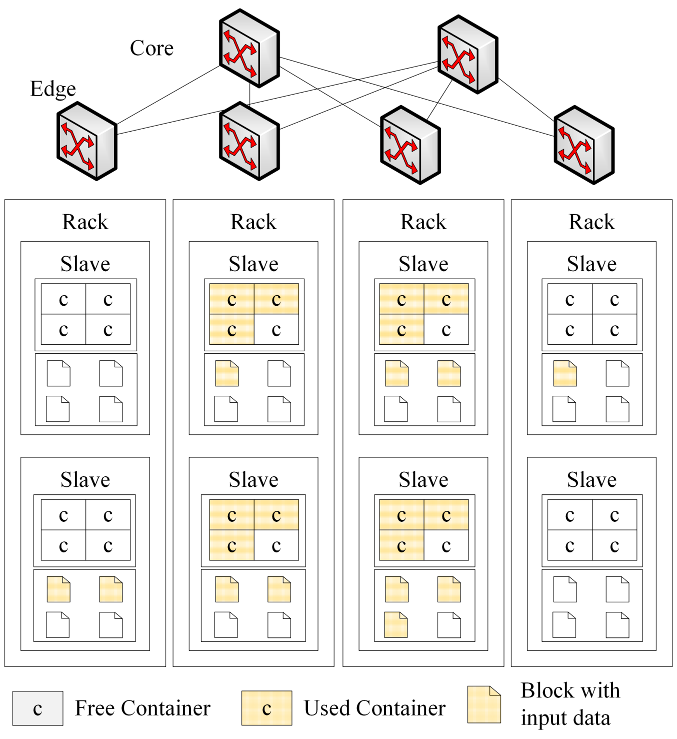

Fig. 6: A container placement with balanced available resource.

V. Evaluation

This section describes the evaluation of our container placement approach on a 8x8 core cluster organized in four racks using a fat tree topology. First, we provide more details about our experimental setup. Afterwards, we present results of four experiments. The experiments we present use two iterative algorithms, K-means clustering and Connected Components, and submit a number of concurrent Flink jobs, reflecting a high and mid cluster utilization.

A. Experimental Setup

We evaluated our scheduling approach on an eight node cluster, in which each node is equipped with eight cores, 32 GB of RAM, a single SATA disk and a 1 Gbps Ethernet network interface. We used an additional node as the master node for the cluster. The eight worker nodes are organized in four racks, so each rack consists of two nodes. The nodes are connected through a fat tree topology with two switching elements on the core and four on the edge layer. All switches (HP ProCurve Switch 1800-24G) are SDN capable and support OpenFlow 1.1.

The single master node manages the data storage and computing functionalities. Therefore, it runs YARN's ResourceManager, HDFS's NameNode and OpenDaylight as a SDN network controller. In addition, our component to calculate a good container placement is hosted here. The remaining eight slave nodes are responsible for storing the data and running the workloads within containers. Therefore, each node runs YARN's NodeManager and HDFS DataNode. In our experiments all workloads run within containers with 1 vcore and 3 GB memory. Thus, we were able to run 64 containers at the same time.

In terms of software all nodes run Ubuntu 14, Apache Hadoop 2.7 and Apache Flink 0.9. We modified the implementation of Flink's application master so it requested placement hints from our placement component before allocating containers. The modification contains only a few lines of code. In particular, the application master makes a request to our placement component and afterwards sends the received host preferences to YARN's ResourceManager.

Since our test bed has only a few nodes, yet our approach targets hierarchical networks with hundreds or thousands of nodes of which network traffic aggregates on the core network, we decided to shape the network interfaces of the core network to simulate our target environment. Assuming 48 port switches available on the rack level with 1 Gbps available per port, we chose a blocking factor of 1:5 [18], and thus have 8 ports with collectively 8 Gbps uplink bandwidth available between the core switches. The remaining 40 ports of the top-of-rack switches are available for hosts. Because we actually have only two nodes per rack, not 40, we shape the link between the top-of-rack switches and core switches to 400 Mbps (8 Gbps divided by 20 as we only a twentieth of nodes). Therefore, in our fat tree topology, each top-of-rack switch has an uplink with 200 Mbps to each of the two core switches.

Cluster Utilization	Workload	Improvements
High (60 of 64 cores)	K-Means	39%
	Connected Components	67%
Mid (36 of 64 cores)	K-Means	40%
	Connected Components	45%

TABLE I: Overview of experiments with four different Flink workloads.

B. Preliminary Results

In this section we show the effects of our placement on workloads consisting of two iterative algorithms, namely K-means and Connected Components.

KMeans Clustering [30] is a compute intensive data processing algorithm, which is used in the area of Machine Learning. It is an iterative algorithm that groups a large set of multi-dimensional data points into k distinct clusters without supervision. For our evaluation, we used five random fixed centers and 600 million points, resulting in around 8 GB input data.

Connected Componenents [31] is an iterative graph algorithm that identifies the maximum cardinality sets of vertices that can reach each other in an undirected graph. For our evaluation, we used a Twitter dataset with around 25 GB input data [32].

High cluster utilization. For the first experiments, we used 60 of 64 available cores. We submitted five applications, each using 12 containers, in which one was used for the application master and eleven for Flink tasks. Each containers allocated 3 GB of RAM and 1 vcore. We run two different workloads under this utilization. One was using the K-means clustering job, the other the Connected Components job. In both workloads, we submitted the five applications with a delay of 10 seconds in-between. For each job we generated a separate dataset to have a different locations for the input data blocks of different applications. Figure 7 shows the results, where each block represents the execution time of an application, stacked to sum the execution time of all five applications. The speedup for the k-means clustering workload was 39% and for the connected components 67%.

Mid cluster utilization. For the second experiments, we used 36 of 64 available cores. In this experiment we submitted three applications, each using 12 containers. Each container allocated 3 GB of RAM and 1 vcore. We run two different workloads under mid utilization. One conisted of K-means clustering jobs and the other of Connected Component jobs. In both workloads, we submitted the applications with a delay of 10 seconds. For each job we generated a separate dataset to have a different locations for the input data blocks of different applications. Figure 8 shows the results, shows the results, where each block represents the execution time of an application, stacked to sum the execution time of all three applications. The speedup for the K-means clustering workload was 40% and for the Connected Components workload 45%.

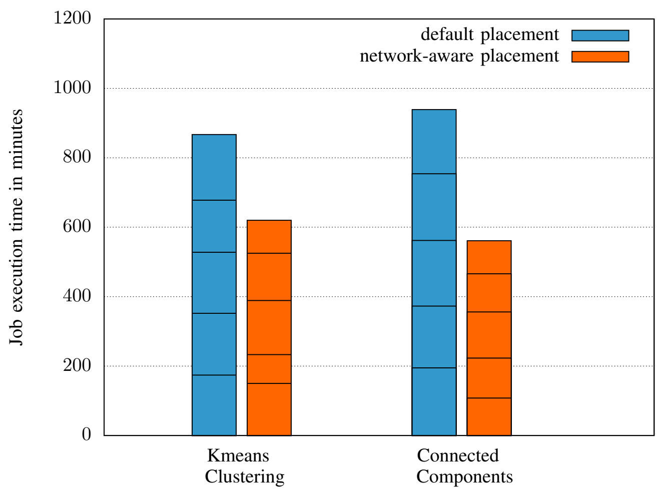

Fig. 7: Workloads consisting of five K-means jobs improved by 39% for k-means clustering, while workloads of five Connected Components jobs improved by 67%.

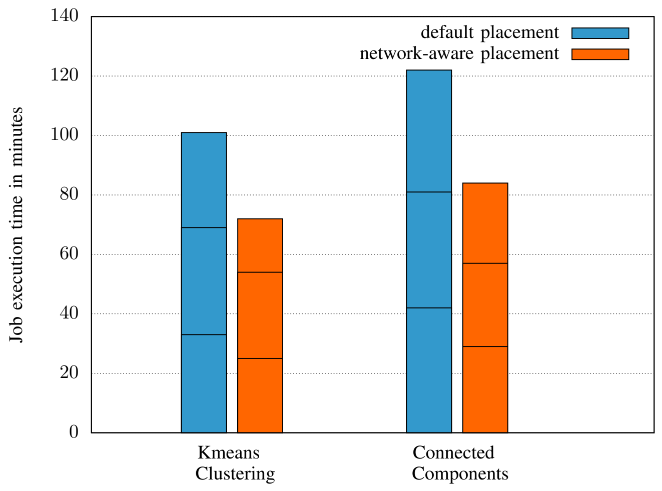

Fig. 8: Workloads consisting of three K-means jobs improved by 40% for k-means clustering, while workloads of three Connected Components jobs improved by 45%.

VI. Related Work

Our approach to container placement draws from a range of existing works. Yet, to the best of our knowledge, no resource management system provides container placement for scalable data analytics frameworks with a focus on hierarchical datacenter networks.

A. *Cluster Scheduling for Data-Analytics Frameworks*

Cluster scheduling for Data-Analytic Frameworks has been an area of active research over the last years. Recent work has been proposed techniques on addressing fairness across multiple tenants [33]–[36], time-predictable resource allocation [10], and improving data locality [11]–[16]. Work on data locality is similar to our solution, as data locality is also part of our cost function. In most of these works, however, data locality stops at rack level and does not take the underlying network topology into account. In addition, most of these works focus on MapReduce and do not support a broader range of data analytics frameworks. Our approach targets the resource management level and container placement in particular. Thus, a number of frameworks benefits from the improvements. Alkaff et. al. [28] present a scheduler that asigns tasks and allocates network paths topology-aware by using Software-Defined Networking. However, they focus on streaming frameworks like Storm [6] and ignore data locality.

B. *Resource Management Systems*

Different resource management system with a focus on big data like YARN [7], Omega [37], or Mesos [8] have been developed. In Mesos or YARN, network bandwidth is not explicitly modeled. In contrast, Oktopus [18] and Orchestra [38] explicitly consider network resources. However, these systems do not primarlily target scalable data analytics.

Mesos helps frameworks to achieve data locality without knowing which nodes store input data for the application. This is realized by offering resources to frameworks and allowing frameworks to reject offers. Thus, a framework can reject offers that do not satisfy data locality thresholds and accept the ones that do. In combination with delay scheduling [11], in which frameworks wait for a limited time to acquire nodes storing the input data, Mesos can archieve good data locality. However, this leaves finding placements with good data locality to the framework. Moreover, to the best of our knowledge, there is no resource management system or scheduling approach available that takes both hierarchical network and data locality into account.

VII. Conclusion and Future Work

There is a strong need to share cluster resources among users and frameworks, since workloads consist of different applications, implemented in different frameworks analyzing different datasets. Typically, many of these applications are network intensive and, thus, container placement for these applications must take network topologies and the locations of input data into account. For this reason, this paper presents an approach and an architecture for effective container allocation for scalable data analysis in shared clusters with hierarchical networks. We implemented our approach on top of YARN and evaluated it with workloads consisting of Flink jobs. Our evaluation showed a reduction of job runtimes of up to 67 %.

In the future, it would be interesting to configure underlying cost function, which is optimized in the process of finding good container placements for applications, automatically depending on factors such as the cluster setup but also frameworks, applications, and datasets. Another option would be to further reduce network congestions in hierarchical multipath networks with SDN-based load balancing for these data analysis frameworks.

Even without these further improvements, we believe this work is already an important step towards making cluster resource management systems more aware of the underlying datacenter network, which becomes more and more important for big data applications as the size of clusters grows.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-parallel Programs from Sequential Building Blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10.

[4] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl *et al.*, "The Stratosphere platform for big data analytics," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 23, no. 6, pp. 939–964, 2014.

[5] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: a timely dataflow system," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 439–455.

[6] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 147–156.

[7] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop YARN: yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[8] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," in *NSDI*, vol. 11, 2011, pp. 22–22.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1–10.

[10] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao, "Reservation-based Scheduling: If You're Late Don't Blame Us!" in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

[11] M. Zaharia, D. Borthakur, J. Sen Satodorma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 265–278.

[12] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, "The power of choice in data-aware cluster scheduling," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 301–316.

[13] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 287–300.

[14] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 261–276.

[15] X. Bu, J. Rao, and C.-z. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 227–238.

[16] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55*, 2009.

[17] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[18] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[19] M. Veiga Neves, C. De Rose, K. Katrinis, and H. Franke, "Pythia: Faster big data in motion through predictive software-defined network optimization at runtime," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, May 2014, pp. 82–90.

[20] C. Fuerst, S. Schmid, L. Suresh, and P. Costa, "Kraken: Towards elastic performance guarantees in multi-tenant data centers," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2015, pp. 433–434.

[21] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985.

[22] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu, "Transparent and flexible network management for big data processing in the cloud," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, 2013.

[23] Apache, "Hadoop," http://hadoop.apache.org (accessed September 2015).

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[25] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, "Spinning fast iterative data flows," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1268–1279, 2012.

[26] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI, "Making sense of performance in data analytics frameworks," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)(Oakland, CA*, 2015, pp. 293–307.

[27] K. Kirkpatrick, "Software-defined networking," *Commun. ACM*, vol. 56, no. 9, pp. 16–19, Sep. 2013.

[28] H. Alkaff, I. Gupta, and L. Leslie, "Cross-layer scheduling in cloud systems," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, March 2015, pp. 236–245.

[29] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke, "Nephele/pacts: A programming model and execution framework for web-scale analytical processing," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 119–130.

[30] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

[31] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Commun. ACM*, vol. 16, no. 6, pp. 372–378, Jun. 1973.

[32] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *WWW '10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 591–600.

[33] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *NSDI*, vol. 11, 2011, pp. 24–24.

[34] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multiresource allocation: Fairness–efficiency tradeoffs in a unifying framework," *Networking, IEEE/ACM Transactions on*, vol. 21, no. 6, pp. 1785–1798, 2013.

[35] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical scheduling for diverse datacenter workloads," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 4.

[36] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: scalable and coordinated scheduling for cloud-scale computing," in *Proc. USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2014.

[37] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, Scalable Schedulers for Large Compute Clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 351–364.

[38] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 98–109.