Improving big data application performance in edge-cloud systems

Dávid Haja[*][†], Balázs Vass[†], László Toka[*][†][‡]
[*]MTA-BME Network Softwarization Research Group [†]Budapest University of Technology and Economics
[‡]MTA-BME Information Systems Research Group

***Abstract*—Data analysis is widely used in all domains of the economy. While the amount of data to process grows, the time criteria and the resource consumption constraints get stricter. These phenomena call for advanced resource orchestration for the big data applications. The challenge is actually even greater at the advent of edge computing: orchestration of big data resources in a hybrid edge-cloud infrastructure is challenging. The difficulty stems from the fact that wide-area networking and all its well-known issues come into play and affect the performance of the application. In this paper we present the steps we made towards network-aware big data application design over such distributed systems. We propose a HDFS block placement algorithm for the network reliability problem we identify in geographically distributed topologies. The heuristic algorithm we propose provides better big data application performance compared to the default block placement method. We implement our solution in our simulation environment and show the improved quality of big data applications.**

***Index Terms*—Big data, edge, reliability, HDFS**

I. Introduction

Edge-cloud computing [1] and mobile edge computing [2] are novel concepts extending traditional cloud computing by deploying compute resources closer to customers and end devices. This approach, closely integrated with carrier-networks, enables several future 5G applications and network services, such as novel Industry 4.0 use-cases, Tactile Internet, or remote driving. Edge resources provide execution environments close to users in terms of latency (e.g., in mobile base stations). By these means, on the one hand, customers' devices can offload computational tasks to this environment instead of consuming their local resources. On the other hand, functions can be transferred from central clouds to the edge enabling latency-critical communication, required by envisioned services. Many of the time-critical applications belong to the domain of data analysis. Data processing is performed in the edge either to provide the fastest possible actuation or for lowering the network resource consumption towards the cloud. In both cases the applications are deployed in a hybrid platform, i.e., using computation and/or storage not only in the edge but also in the cloud. In these cases however the performance of the wide-area network greatly affects the quality and the efficiency of the big data operation.

Today's resource orchestration algorithms, used for big data application scheduling, are usually designed for a single data center. Consequently they do not consider the characteristics of the underlying network, as reliability, latency and bandwidth-capacity rarely pose problems in the typically dense data center networks. However, these network aspects can easily cause application performance degradation when the compute infrastructure contains edge nodes inter-connected with wide-area networking. The lack of sheer connectivity may cause problems for the most widely used big data applications, even though those were designed with robustness and redundancy against node failures. We argue that this networking aspect must be taken into account by a big data system designer, particularly in a geographically distributed edge-cloud scenario.

In this work, we engage in this problem domain and propose a HDFS block placement method that can significantly improve the system performance compared to the default baseline. We propose a fast topology-aware method for HDFS block placement in order to gain a 6-fold increase in HDFS availability. Our general finding is that the widely used open-source big data frameworks are not yet ready for hybrid edge-cloud deployments where the quality of wide-area networks might leave a huge mark on the application performance. Our work is a pioneer step towards geo-distributed network-aware orchestration of such systems: to the best of our knowledge, these aspects have never been investigated in relation to big data and edge-cloud infrastructure.

II. HDFS block placement to increase data availability

In this section we present how we can increase the availability of HDFS by carefully selecting the servers that host HDFS blocks, based on the edge-cloud topology. We prove that this is in fact a hard problem, and we show a fast heuristic method that increases the block availability compared to what the default HDFS algorithm achieves.

A. The HDFS block placement problem

Our proposed solution modifies the default rack-awareness of HDFS with the ability of choosing not only from server racks, but also among clusters. Our goal is to place HDFS block replicas on the servers that are the most reliably reachable from the given vantage point, the HDFS client.

We model the edge-cloud topology with a graph in which a vertex is one server, switch or gateway. We assume two special vertices in the graph: a central vertex that represents the Internet, and a client vertex from where the HDFS data is being reached. The servers are grouped into server racks which are further grouped into clusters. We distinguish two types of clusters: data centers and edge clusters. Each cluster contains at least one gateway node that provides the connection between the cluster and the Internet. The nodes of the graph are connected by undirected edges that correspond to physical links. Each edge e is associated with a positive p_e value that represents the failure probability of the edge in a given time frame. We consider that links fail independently of each other. The failure value of a path between two arbitrary vertices, i.e., the probability of the path to break, is calculated based the p values of the path's edges. Let us have a path from source vertex u to destination server v, containing only two edges (denoted by u and v after their incident vertices respectively); in this case the server's reliability is equal to $(1-p_u)(1-p_v) = 1 - p_u - p_v + p_u p_v$, where if p_u and p_v are small enough, then $p_u p_v$ is negligible. Generally, a server's reliability can be approximated with $1 - \sum_{u,v \in V} p_{u,v}$, where $p_{u,v}$ gives the failure probability between u and v neighbor vertices, for every edge in the path. Our goal is to maximize this value to achieve better reliability for accessing a server.

Let R denote the replication factor of HDFS blocks. We argue that one should select servers that are reachable through edge-disjoint paths.

Lemma 2.1: If the edge failure probability is sufficiently low, then a collection of R edge-disjoint paths always provides higher reliability than R non-edge-disjoint paths.

Proof: Let us look for R vertices $v_1, \ldots, v_R$ in the graph. For simplicity, suppose that p is the same for every edge. The probability of an overall failure of all the independent paths, leading from the common vantage point to each of the $v_1, \ldots, v_R$ vertices, for small p values can be approximated by $p^R \prod_{i=1}^{R} l_i$, where $l_1, \ldots, l_R$ are the lengths of the paths to those vertices. Note that $p^R \prod_{i=1}^{R} l_i$ can be upper bounded by $p^R E^R$, where E is the total number of edges in the graph. On the other hand, if there exists two paths having at least one shared edge, the common failure probability is lower bounded by p^{R-1}. Based on these two bounds, in case $p^R E^R < p^{R-1}$ stands, or equivalently $p < \frac{1}{E^R}$, i.e., the p value is small enough which is usually the case in real networks, the edge-disjoint path collections are more reliable than non-edge disjoint path collections. ■

So if p is small enough, our task is to find disjoint paths in a general graph. This is an NP-complete problem.

Lemma 2.2: In a highly reliable infrastructure the HDFS block placement problem is NP-complete.

Proof: Based on disjoint path search in a general graph, one can prove that there are instance families of our problem in which finding the safest placement is at least as hard as deciding whether feasible simple integer flows exist in the general graph, which is a known NP-complete problem (Thm. 4 of [8]). The key of the proof is to extend the network topology with an imaginary target node t, which is connected to all the possible locations, i.e., $v_1, \ldots, v_R$. ■

B. Our failure-aware algorithm for HDFS block placement

In this subsection we present our network failure-aware heuristic algorithm, which places the HDFS blocks into the edge-cloud topology.

In the initial phase the algorithm determines the reliability of each server: it calculates the shortest path in terms of edge failure probabilities from the vantage vertex to each server rack (as all servers in a rack are reached on the same path with the same reliability). The complexity of finding the shortest path to all server racks is $\mathcal{O}(E + V \log V)$, where V is the number of vertices and E is the number of edges in the graph. Then the algorithm sorts the servers in each cluster separately by their assigned reliability values. The sorting can be done in $\mathcal{O}(V \log V)$. The complexity of the whole initial phase is therefore $\mathcal{O}(E + V \log V)$.

Algorithm 1 Cluster selection for HDFS block replicas

```
1:  if $replication\_factor > 2$ then
2:      $two\_replica\_in\_one\_DC \leftarrow False$
3:      for ($i = 0; i < replication\_factor$; i++) do
4:          $ordered\_cluster\_list \leftarrow$ order_clusters_by_utilization($topology$)
5:          while $i < 2$ do
6:              if $any(cluster$ in $ordered\_cluster\_list$ where $cluster.type! = "edge"$ and $cluster.available\_racks > 1)$ and $two\_replica\_in\_one\_DC$ is $False$ then
7:                  $chosen\_cluster \leftarrow$first cluster from $ordered\_cluster\_list$ where $cluster.type! = "edge"$ and $cluster.available\_racks > 1$
8:                  $two\_replica\_in\_one\_DC \leftarrow True$
9:              else
10:                 if $any(cluster$ in $ordered\_cluster\_list$ where $cluster.type! = "edge"$ and $cluster$ not used then
11:                     $chosen\_cluster \leftarrow$first cluster from $ordered\_cluster\_list$ where $cluster.type! = "edge"$ and $cluster$ not used
12:                 else
13:                     if $any(cluster$ in $ordered\_cluster\_list$ where $cluster$ not used then
14:                         $chosen\_cluster \leftarrow$first cluster from $ordered\_cluster\_list$ where $cluster$ not used
15:                     else
16:                         $chosen\_cluster \leftarrow$first cluster from $ordered\_cluster\_list$
17:                     end if
18:                 end if
19:             end if
20:         end while
21:         if $any(cluster$ in $ordered\_cluster\_list$ where $cluster.type == "edge"$ and $cluster$ not used) then
22:             $chosen\_cluster \leftarrow$first cluster from $ordered\_cluster\_list$ where $cluster.type == "edge"$ and $cluster$ not used
23:         else
24:             if $any(cluster$ in $ordered\_cluster\_list$ where $cluster$ not used) then
25:                 $chosen\_cluster \leftarrow$first cluster from $ordered\_cluster\_list$ where $cluster$ not used
26:             else
27:                 $chosen\_cluster \leftarrow$first cluster from $ordered\_cluster\_list$
28:             end if
29:         end if
30:     end for
31: else
32:     for ($i = 0; i < replication\_factor$; i++) do
33:         $chosen\_cluster \leftarrow (ordered\_cluster\_list[i])$
34:     end for
35: end if
```

Alg. 1 describes how our algorithm selects clusters for the replicas. First, it sorts the clusters by their utilization; after that if the replication factor is less than three, it puts one replica in the least utilized cluster and another one into the second least utilized cluster. If the replication factor is greater or equal to three, it tries to place the first two replicas following the policy that prioritizes data centers against edge-clouds: 1) Place them in a data center where two racks have available free space; 2) Place them in two separate data centers; 3) Place them in two separate clusters; 4) Place them in the least utilized cluster.

In the chosen cluster(s) the algorithm disperses the first two replicas to different racks, chosen based on their decreasing order of reliability. Further replicas will be dispersed among edge clusters. During this second phase, in the worst case, the selection of racks iterates through all the vertices four times for each replica (see Alg. 1 for details). So the overall complexity of our HDFS block placement algorithm is $\mathcal{O}(E+V(\log V+R))$, where R is the replication factor.

C. Simulation settings and results

We evaluated our heuristic algorithm against default HDFS block placement method in case of network failures. In simulations we used two data center topologies: CLOS [7] and Fat Tree [6]. Each topology had 50 edge clusters and 5 data centers. We assumed that each edge had "four nines" availability in a data center, while links in the edge clusters had only "three nines" availability, i.e., the edge cluster links are ten times more likely to fail than the data center's. An edge failure might be due to the failure of a device and/or a physical link, and also due to misconfigurations, power outages, etc. In each simulation the default HDFS replication factor value (3) was used. In order to evaluate the effect of network failures on HDFS, a simulation first fills the servers with HDFS blocks, then randomly removes edges from the topology until data loss happens. Data loss is when all of the replicas belonging to one HDFS block are unreachable from the HDFS client vertex, our vantage point. We recorded the number of unavailable vertices when the data loss happened: a vertex is considered unavailable if it is unreachable from the vantage vertex.

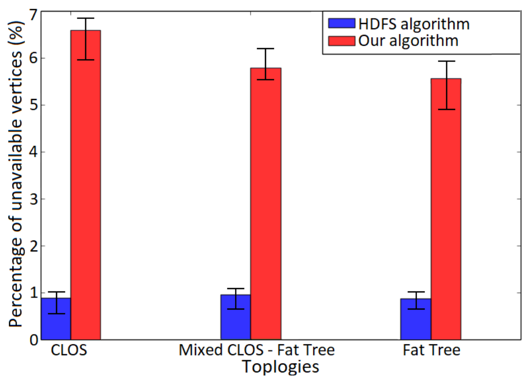

Fig. 1. HDFS block availability simulation results at the first failure

In Fig. 1 we show the results: in each bar group (representing different data center topologies) the bars depict the percentage of unavailable vertices when the first HDFS block's replicas become all unavailable. The left blue bar shows the result given by the HDFS algorithm, and the right red bar shows the results given by our algorithm. In leftmost bar group results are shown on CLOS data center topologies; in the middle, simulations assumed both CLOS and Fat Tree data centers; in the rightmost scenario only Fat Tree topologies were used. In general, given these topologies, it is enough to lose about one percent of nodes in order to lose data using the default algorithm of HDFS. When using our replica placement algorithm, it takes about six times as many nodes to experience data loss. It can be seen that our algorithm performed best in the topology, where there were only CLOS data centers. This result is due to the fact that in the CLOS topology each vertex has a higher degree than nodes in Fat Tree data centers.

III. Conclusion and Future Work

Network reliability is a classic topic in network research. There are many research efforts dealing with this area [3]–[5]. To the best of our knowledge, no one has examined HDFS block placement taking into account network reliability.

In the presented research we investigate the performance degradation of big data applications once they are deployed in a geographically distributed infrastructure. As edge computing becomes rather the norm than the exception nowadays, the long-researched networking aspects play an important role when it comes to the availability or the quality of data analysis frameworks. We showed that the default scheduling and resource orchestration strategies of the mainstream big data ecosystems are not prepared to cope with the challenges the wide-area networks pose. Realizing this, we made pioneer steps and proposed a fast topology-aware algorithm for HDFS in order to improve block availability. We have shown in numerical simulations that greedy heuristics yield significant amelioration compared to the default baseline in this aspect.

We plan to extend the open source block placement solution of HDFS with the proposed algorithm and share the code with the respective Apache project along with the extension that manually receives or automatically discovers the underlying topology. The theoretical way forward is to further polish the algorithm, particularly on the basis of feedback from the real-world deployments' underlying topologies.

Acknowledgment

This work was supported by the National Research, Development and Innovation Office (NKFIH) under the research and development project in Hungarian-Korean cooperation (project identifier: 2018-2.1.17-TT-KR-2018-00012).

References

[1] Charles C. Byers, "Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks", IEEE Communications Magazine, 2017.
[2] P. Mach *et al.*, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading", IEEE Communications Surveys and Tutorials, 2017.
[3] J. Meza, *et al.*, "A Large Scale Study of Data Center Network Reliability", ACM IMC, 2018.
[4] C. J. Colbourn, "The Combinatorics of Network Reliability", "Oxford University Press, Inc.", 1987.
[5] I. B. Gertsbakh, *et al.*, "Models of Network Reliability: Analysis, Combinatorics, and Monte Carlo", "Boca Raton: CRC Press.", 2012.
[6] M. Al-Fares, *et al.*, "A scalable, commodity data center network architecture", ACM SIGCOMM, 2008.
[7] C. Clos, "A study of non-blocking switching networks", IEEE The Bell System Technical Journal, 1953.
[8] S. Even, *et al.*, "On the complexity of timetable and multi-commodity flow problems", SIAM Journal on Computing, 1976.