Scientific workflow management on hybrid clouds with cloud bursting and transparent data access

Bartosz Baliś[1][0000−0002−3082−4209], Michał Orzechowski[1], Łukasz Dutka[2][0000−0002−2198−6556], Renata G. Słota[1][0000−0001−7424−9317], and Jacek Kitowski[1,2][0000−0003−3902−8310]

[1]AGH University of Science and Technology, Department of Computer Science, Krakow, Poland
[2]AGH University of Science and Technology, ACK Cyfronet AGH, Krakow, Poland
{morzech, balis, rena, kito}@agh.edu.pl

Abstract. Cloud bursting is an application deployment model wherein additional computing resources are provisioned from public clouds in cases where local resources are not sufficient, e.g. during peak demand periods. We propose and experimentally evaluate a cloud-bursting solution for scientific workflows. Our solution is *portable* thanks to using Kubernetes for deployment of the workflow management system and computing clusters in multiple clouds. We also introduce *transparent data access* by employing a virtual distributed file system across the clouds, allowing jobs to use a POSIX file system interface, while hiding data transfer between clouds. To *balance load distribution* and *minimize the communication volume* between clouds, we leverage graph partitioning, while ensuring that the algorithm distributes the load equally at each parallel execution stage of a workflow. The solution is experimentally evaluated using the HyperFlow workflow management system integrated with the Onedata data management platform, deployed in our on-premise cloud in Cyfronet AGH and in the Google Cloud.

Keywords: cloud bursting, scientific workflow management, scientific data management, hybrid cloud, Kubernetes

1 Introduction

Cloud bursting is an application deployment model leveraging *hybrid cloud* [17], in which the application normally runs in a private cloud, while during peak demand periods it is partially deployed on additional resources allocated in a public cloud [10]. Hybrid computing approach based on on-premise computing infrastructure and resources allocated on-demand from a public cloud is increasingly adopted in scientific computing [19]. The motivations to move scientific computations to public clouds include lack of appropriate local resources [4], insufficient computing infrastructure due to growing user base [1], and better turnaround times achieved

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

in the cloud due to long queue wait time in a local cluster, even if it is better in terms of raw performance [3,16]. Consequently, cloud bursting is definitely an attractive model for scientific applications.

In this paper, we propose a cloud bursting approach for scientific workflows leveraging graph partitioning. In our solution we address the following problems: (1) *Portability*: we deploy the Workflow Management System and computing clusters on the private and public clouds using Kubernetes, taking benefit from its cloud-agnostic application deployment model [20]. Consequently, we support any cloud in which a Kubernetes cluster can be deployed. (2) *Transparent data access*: we employ a virtual distributed file system across two clouds, so that workflow jobs use a POSIX file system interface, while data is transparently transferred regardless of its location (local or remote), and underlying storage technology. (3) *Execution optimizations*: we introduce an algorithm for partitioning of the workflow in order to balance task distribution between partitions (load balancing in space) and in individual parallel execution phases of the workflow (load balancing in time); the algorithm allows uneven partitions (cluster sizes), and optimizes the volume of communication between clouds. We estimate the required size of the cluster rented in the public cloud by taking into account the level of parallelism of the workflow and resource demands of workflow jobs. The solution is experimentally evaluated with the HyperFlow workflow management system [2] integrated with the Onedata data management platform [8].

The paper is organized as follows. Section 2 presents related work. Section 3 describes the proposed solution. Section 4 contains experimental evaluation of the solution. Section 5 concludes the paper.

2 Related Work

The usefulness of hybrid infrastructures in scientific computing have been recognized since the early days of clouds [5, 21, 24]. Hybrid clouds have also been the subject of research specifically in the context of scientific workflows. Liu and others [15] introduce an algorithm for dynamic placement of large scientific workflow data sets in hybrid clouds. In [13] and [6], the authors propose algorithms for scheduling of workflows on hybrid clouds, with deadline and cost constraints. These algorithms heavily rely on accurate predictions of task execution times on specific computing resources which can be difficult [23].

In [22] graph partitioning is used to guide scheduling of scientific workflows in a cluster of nodes, but not in the context of a hybrid cloud. Cloud bursting is supported in the Galaxy system dedicated for life science workflows [9]. The key component in this solution is a job mapper which decides to which cloud a given job should be submitted. This decision-making is based on simple criteria, e.g. the resource availability, location of specific tools, or location of data [1].

Overall, no approaches investigate graph partitioning for hybrid execution of scientific workflows with cloud bursting, combined with architectural aspects enabling portability and transparent data access. All these issues are addressed by the research presented in this paper.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

3 Cloud bursting solution for scientific workflows

3.1 Architecture

Fig. 1 shows a conceptual architecture of the proposed solution. The private cloud is running Kubernetes cluster 1 – the *home cluster*. We assume that the user is assigned a certain quota of N virtual CPUs (vCPU) in the home cluster, distributed across a certain number of nodes. The user decides to allocate another Kubernetes cluster in a public cloud in order to speed up the computations. The second cluster has size of M vCPUs.

The workflow graph is then divided into two partitions, one for each of the clusters. The relative sizes of the partitions are, respectively, $\frac{N}{N+M}$ and $\frac{M}{N+M}$. The partitioning should minimize the volume of the inter-cluster communication and maximize task parallelism, see section 3.2 for more details.

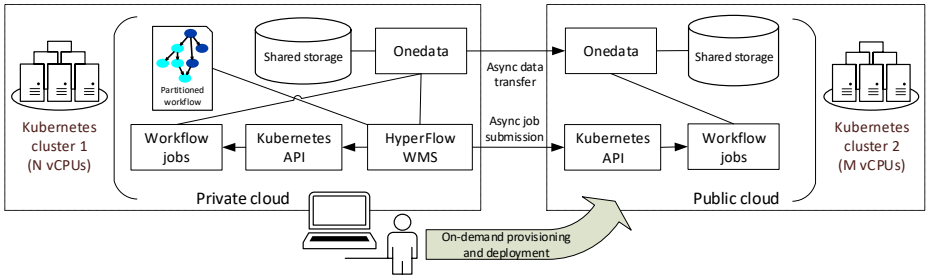


Fig. 1: High-level concept of the proposed cloud bursting architecture for scientific workflows. The user has a cluster with N virtual CPUs in the private cloud and allocates another cluster with M vCPUs in the public cloud. The workflow graph is divided into two partitions proportional to cluster sizes. The partitioning should minimize inter-cloud data transfer without reducing task parallelism.

How to estimate the desired size (in virtual CPUs) of the second cluster? The upper limit on M can be estimated as follows:

$$M = \sum_{p_i \in P_{LoP}(w)} cpuReq(p_i) - N_{vcpu}(cluster_1) + 0.5 * N_{nodes} \quad (1)$$

where $P_{LoP}(w)$ is the subset of workflow tasks that belong to the largest graph level (execution phase) of the workflow graph. In other words, a cluster of its size would accommodate for the largest level of parallelism (LoP) in the workflow and avoid task waits. The factor $0.5 * N_{nodes}$ represents the fact that about 0.5 of vCPU is reserved on each node for the Kubernetes middleware. Let us note that the Level of Parallelism in a Directed Acyclic Graph (DAG) of tasks is not necessarily equal to the size of the largest level graph. A better estimation can be done by simulating the workflow 'execution wave' [12]. $cpuReq(p_i)$ denotes

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

the CPU request (in vCPUs) of task p_i, while $N_{vcpu}(cluster_1)$ is the number of vCPUs in Cluster 1.

Fig. 1 also shows a simplified deployment of components on both clusters. The HyperFlow engine [2] runs in the home cluster and executes the workflow. Based on the information on graph partitioning, HyperFlow creates workflow jobs either in the home or the remote Kubernetes cluster. The Onedata data management platform [8] is responsible for synchronizing data between the clusters. The input data of the workflow is initially located on the storage node in the home cluster. On the remote cluster, Onedata creates a virtual POSIX file system and synchronizes metadata about the input files, so that the files are visible locally, even though they physically exist in a remote file system. Onedata transfers the files only when they are accessed, and only these blocks that were actually read which is efficient for very large files whose only small parts are read by remote processes.

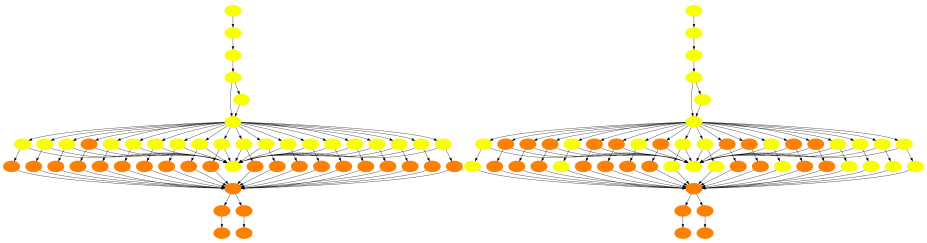
3.2 Workflow partitioning

To map workflow tasks to different clusters (clouds), we use graph partitioning [18]. Graph vertices represent jobs while edges denote communications between them. Vertex *weight* denotes the *computational cost* of a job, in our case the requested amount of *vCPU*, e.g. 0.5. Vertex *size*, on the other hand, represents the *communication cost* due to the execution of the job, i.e., transfer of input files from a storage node and transfer of output files to the storage node. A typical partitioning algorithm will try to minimize the *edge cut* metric, i.e. the number of edges that connect different partitions, while balancing the load (vertex weights) between the partitions. Fig. 2a shows an example workflow partitioned in this way which results in a suboptimal mapping of workflow tasks onto clusters. First, because the volume of communication between clusters is not minimized. Second, because job parallelism is reduced – jobs run only in the first cluster, then only in the second one. In the case of scientific workflows, it is important to divide the computational tasks equally between the partitions at each individual *execution phase* of the workflow [22]. The execution phase of a task $Ep(p_i)$ corresponds to its level in the graph, i.e. it is an integer $1..L$ defined as follows:

- If *task* p_i has no *predecessors*, $Ep(p_i) = 1$.
- Otherwise $Ep(p_i) = max([Ep(predecesors(p_i)]) + 1$.

While the edge cut metric often minimizes the communication volume, for certain types of graphs this is not the case. For example, if a vertex has multiple edges that connect to vertices from another partition, the inter-partition communication would occur once, but the edge cut metric would count each edge separately. For such graphs, a metric optimizing the number of boundary vertices works better, because it actually optimizes the communication volume [18]. Fig. 2b shows a partitioning of the same workflow according to this metric. This partitioning not only reduces the inter-partition communication, but also results in a better task parallelism. As shown in the figure, the two parallel phases of the workflow are distributed across two partitions.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21



(a) Minimizing the edgecut. (b) Minimizing the communication volume.

Fig. 2: Workflow partitioning between clusters. Colors denote partitions to which tasks are assigned.

In our case, we assume that the input data of the workflow is located on a storage node in the home cluster. Therefore, we add an additional, special *storage node* to the graph that has weight of 0 (no computational cost) and represents the transfer of workflow input data to workflow jobs. The storage node is a special *fixed node* which is always pre-assigned to the the partition that represents the cluster where the data is located.

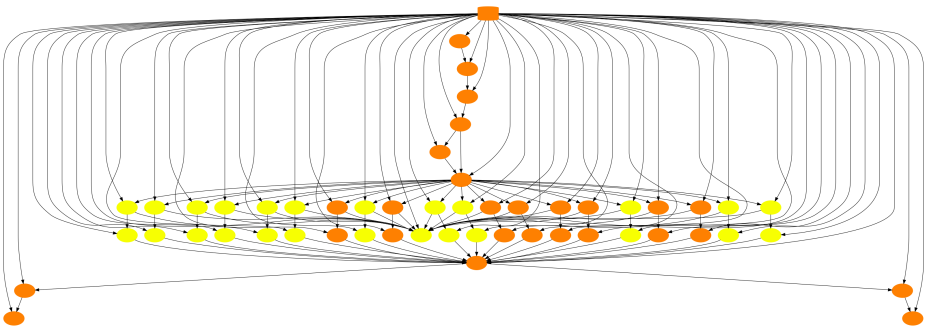


Fig. 3: Workflow graph with the special storage node and unequal partitions (cluster sizes): cluster 1 = 0.4, cluster 2 = 0.6 $*$ total $vCPU$. Partitioning optimized for total communication volume. Colors denote partitions to which tasks are assigned.

A partitioning must also address the problem of different cluster sizes and allow unequal partitions while minimizing communication volume and maximizing task parallelism. Fig. 3 shows a partitioning where the storage node is depicted and the sizes of the partitions vary, such that partition 1 has weight of 0.4, while partition 2 of 0.6.

While we found that the communication volume metric often results in desirable task parallelism, the addition of the storage node may easily disrupt it. Therefore we have decided to use multiple weights for vertices to ensure that the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

load is equally distributed at each parallel execution phase of the workflow [22]. A vertex has now n weights, where n is equal to the number of levels in the workflow graph. For the sake of example, let us assume that a workflow graph has 7 levels. Job p, which belongs to graph level 4, and whose computational cost $C_p = 0.5\ vCPU$ will have the following weights vector:

w_p = [0 0 0 0.5 0 0 0]

In other words, $w_i = C_p$ for $i = level(p)$, 0 otherwise. As a result, the partitioning algorithm will perform multi-constraint optimization, trying to balance each of the weights separately, leading to balanced task parallelism.

4 Evaluation

This section presents evaluation of the proposed solution. Section 4.1 presents analysis of the workflow partitioning algorithm for different synthetic workflow graphs, while section 4.2 describes the setup of the experimental evaluation and shows its results.

4.1 Workflow partitioning analysis

In order to study the impact of different parameters of the algorithm on the partitioning quality, we have analyzed a number of workflow graphs from the Pegasus workflow gallery [7]: Cybershake 1000 vertices, Epigenomics 997 vertices, LIGO 1000 vertices, Sipht 968 vertices, and Montage 2.0 6448 vertices. Fig. 4 presents the results of this analysis, depicting total communication volume of the workflow graph partitioning depending on two variables:

- *distribution of partition sizes*: the value on the x axis denotes the size of the first partition in relation to the entire graph.
- *allowed load imbalance between partitions* (ufactor): 1%, 5%, and 10%.

In addition, the *round shape* of the marker denotes that the storage node is assigned to the bigger partition, while the *diamond shape* means that it is in the smaller partition.

Several observations can be made from these results. In general, not surprisingly, allowing load imbalance among the partitions results in a better communication volume. However, the gain heavily depends on the workflow structure. For Cybershake, for example, it is not significant in any of the partitioning variants, while in the case of Montage 2.0, the difference can be substantial.

Table 1 shows the distribution of workflow tasks between partitions individually for workflow execution phases and for various workflow examples. In all cases partitions sizes were equal and the partitioning was optimized for the communication volume, except for Sipht where the result for the edge cut optimization is also shown for comparison. As one can see, the optimization for the communication volume results in approximately equal distribution of tasks

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

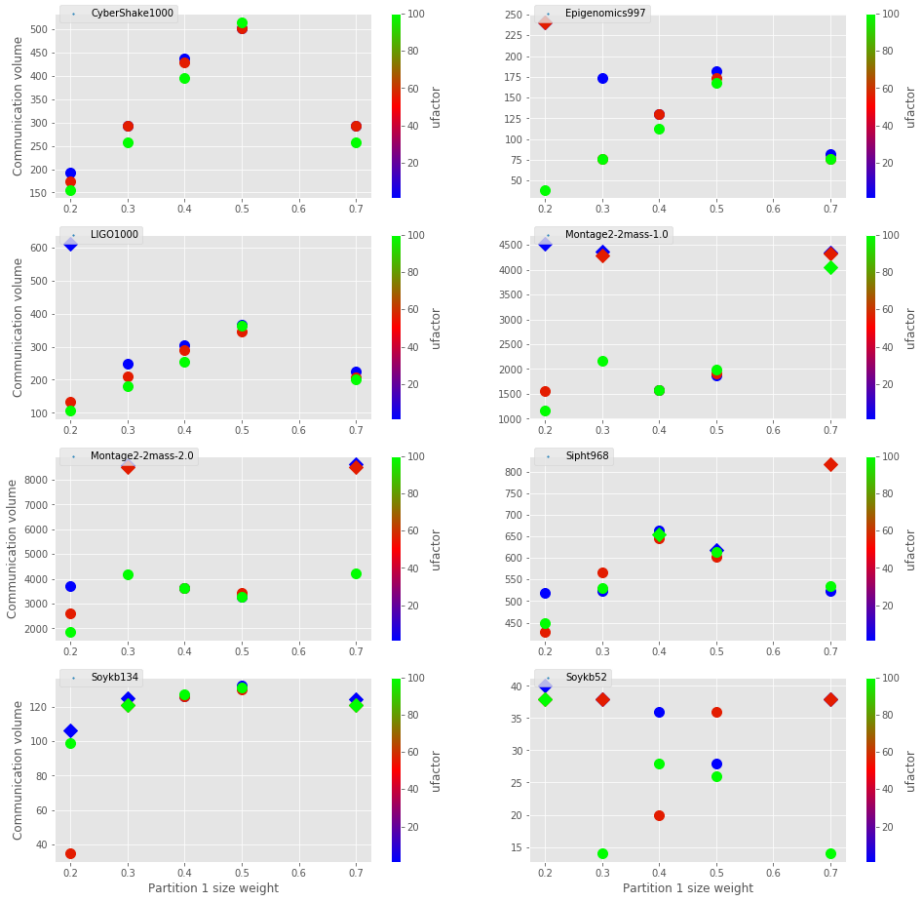


Fig. 4: Workflow partitioning results for eight different workflow graphs showing communication volume for different partition size distribution (0.2/0.8, 0.3/0.7, 0.4/0.6, 0.5/0.5 and 0.7/0.3 and allowed load imbalance between partitions (1%, 5%, 10%, denoted by parameter $ufactor$ of value 10, 50 and 100, respectively).

among partitions, but some divergences are inevitable due to the structure of the workflows, as is in the case of the Montage 2.0 workflow which is perfectly divided when using 3 partitions.

4.2 Experimental runs

For the purpose of experimental runs, we have set up two small Kubernetes clusters: one in the home cloud in Cyfronet AGH consisting of 4 nodes with 6 vCPU and 21 GB of RAM each, and a remote one in Google Cloud comprising 6 nodes, each with 4 vCPU and 16 GB of RAM. Consequently, both clusters were approximately equal in terms of the number of virtual CPUs and the amount of

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

Table 1: Distribution of workflow tasks between partitions for individual execution phases and for different partitioning strategies.

Workflow	**# Tasks**	**Ph1**	**Ph2**	**Ph3**	**Ph4**	**Ph5**	**Ph6**	**Ph7**	**Ph8**	**Ph9**
Cybershake	1000	5/5	240/ 256	240/ 254						
Epigenomics	997	3/4	119/ 126	119/ 126	119/ 126	119/ 126	3/4	1/0	1/0	1/0
Sipht	968	353/ 359	44/20	68/60	21/11	12/20				
Sipht (edge cut)	968	265/ 447	41/23	128/0	32/0	32/0				
Ligo	1000	123/ 106	123/ 106	6/14	123/ 128	123/ 128	6/14			
Montage 2.0 (dss)	6448	102/ 90	3148/ 2900	2/1	1/2	64/ 128	1/2	1/2	1/3	
Montage 2.0 (dss), 3 partitions	6448	64/ 64/ 64	2016/ 2016/ 2016	1/1/1	1/1/1	64/ 64/ 64	1/1/1	1/1/1	2/1/1	

RAM per vCPU. For shared storage, we used NFS in the home cluster and Ceph in the remote one. In both cases, these served as a backed storage for Onedata that exposed a POSIX virtual file system for the workflow jobs.

In the experimental runs we used the genomic workflow Soykb [14], whose structure is shown in Fig. 5. The workflow is characterized by a large input file – the reference genome database, two parallel stages (*genotype_gvcfs* and *filtering_snp* tasks), and a long final task that merges the outputs of previous tasks (*merge_gcvf*). For the purpose of illustration, a small Soykb workflow consisting of 52 tasks is shown, with a relatively small input genome size (about 1 GB). However, genome analysis workflows (including Soykb) can be much larger in size and data footprint [11]. The workflow was divided into equal partitions (with respect to task CPU requests), in a similar way as shown in Fig. 2b.

The visualization of the execution of the workflow in the hybrid cloud is depicted in Fig. 6. Labels on the Y axis denote nodes on which tasks were executed, with `k8s*` / `gke*` denoting nodes in Cyfronet / Google Cloud, respectively. A given node may occur multiple times (e.g. `k8s3-0` and `k8s3-1`) if tasks were running in parallel on this node. Note that in general more tasks were running in parallel on nodes in Cyfronet since they contained more virtual CPUs per node.

Fig. 7 shows the data distribution among the two experimental clouds after the execution of the workflow. As one can see, only about 19% of all input and output files of the workflow were either transferred to or created in the remote (Google) cloud. On the other hand, almost all files were needed in the home cloud, where the final merge tasks were executed. Fig. 7b shows that only about 50% of the largest input file was transferred from the home cloud to the remote

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

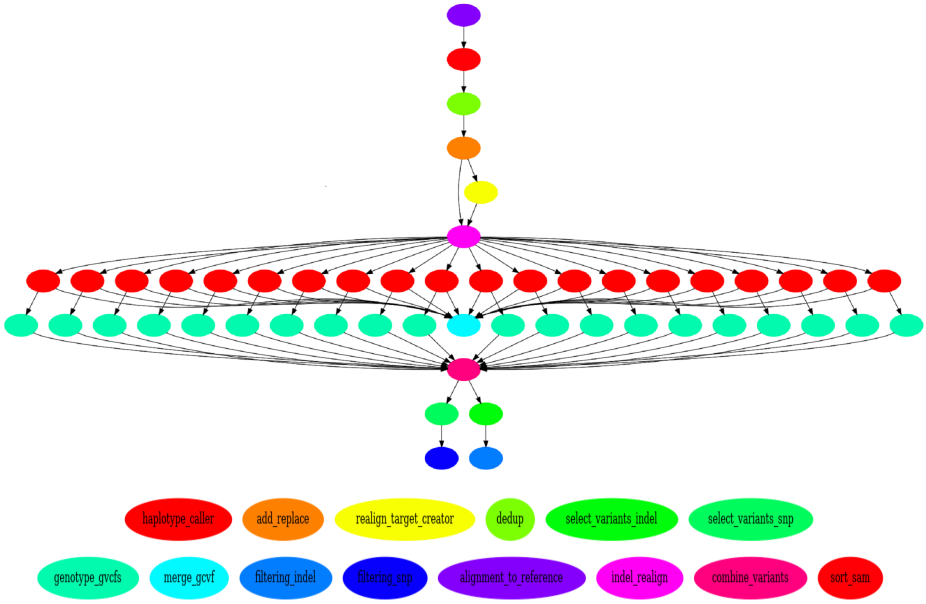


Fig. 5: Structure of the Soykb workflow.

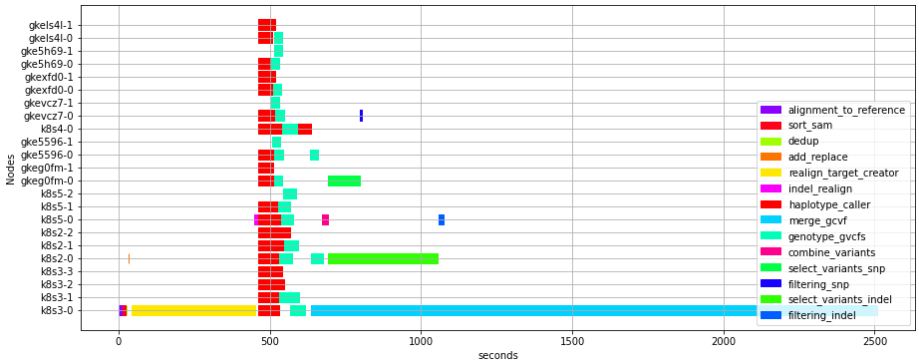
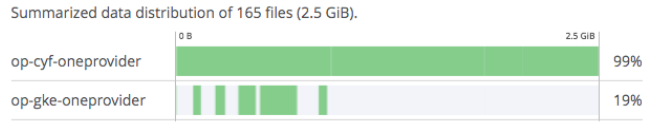


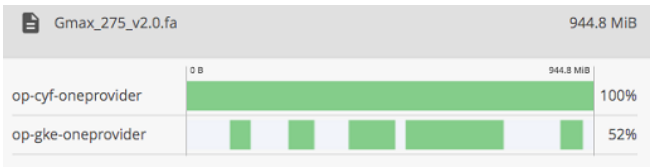
Fig. 6: Example visualization of the Soykb workflow execution in two clouds.

cloud during the execution. This can be explained by looking at the data access pattern of this file, shown in Fig. 8. The chart shows which blocks of the file are accessed by individual jobs of the workflow. The type of jobs is denoted by different colors. The graph reveals that the tasks from the two parallel stages of the workflow (*genotype_gvcfs* and *filtering_snp*) read only small portions of the genome database, clearly showing a data-parallelism pattern. This is an opportunity for optimization since it is not necessary to transfer the entire file to the remote cloud. On the other hand, the two tasks with the 'gather' pattern

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21



(a) All files.



(b) Largest input file.

Fig. 7: Runtime data distribution between two clouds for the Soykb workflow.

(`merge_gcvf` and `combine_variants`) read the entire file which hints that these tasks should run in the home cluster in order to avoid the transfer of the entire file. To guide the partitioning algorithm into this optimization, the weights of the graph edges – denoting the cost of communication – should be based on the **number of file blocks read, not the entire input file size**. However, collecting such data on low-level data access patterns is not always straightforward, requiring instrumentation of low-level file IO subroutines. Because the Onedata system employs block-level data transfer optimization, and the load is divided equally into the two clouds, it is expected that about half of the input file will be transferred to the remote cloud.

4.3 Discussion

The presented method has several limitations. The graph partitioning package that we have used (Metis) does not allow for defining fixed nodes. As a result, the obtained communication volume is only reliable when we assume that the partition to which the storage node was assigned represents the home cluster. Consequently, it could be difficult to optimize partitioning where the home cluster is significantly smaller (e.g. 0.2/0.8) because the storage node may tend to be assigned to the larger partition, depending on the workflow structure.

The method could be improved by introducing information about scheduling of jobs in which their allocation to time slots is taken into account. Such a scheduling algorithm could, rather than mapping jobs to a fixed set of nodes, predict the requirements for resources (vCPUs) in a given point of workflow execution. Consequently, the required size of the public cluster could be estimated better. Moreover, combined with auto-scaling capabilities, the remote cluster could be scaled up and down as needed.

We have focused on balancing the load due to requested vCPUs. The method could be extended to take into account also other resources, e.g. memory and storage. Supporting balancing of memory consumption would be relatively easy to

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

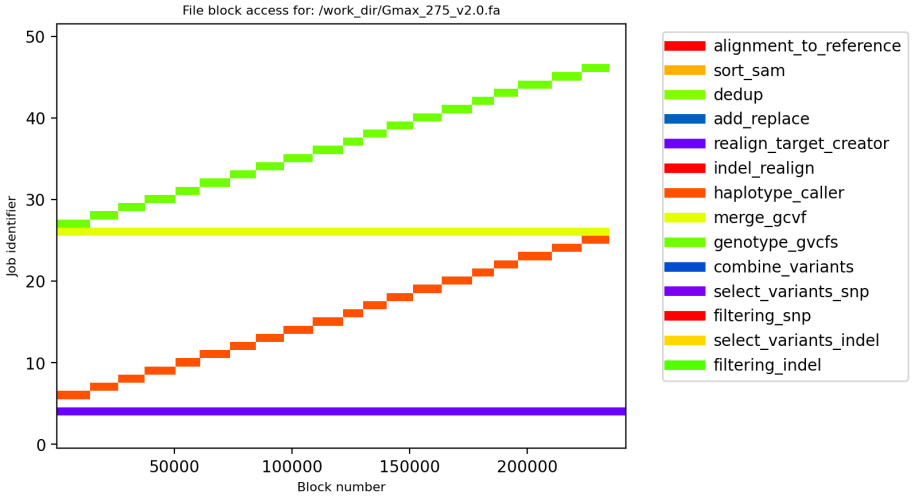


Fig. 8: Data access pattern of the largest input file of the Soykb workflow. The graph shows which blocks of the file were read by individual jobs with colors denoting different job types. The pattern reveals that the tasks from the two parallel stages of the workflow (*genotype_gvcfs* and *filtering_snp*) read only small portions of the file.

implement – it would require adding memory-request weights to the partitioning algorithm. The Kubernetes scheduler already supports memory requests and assigns jobs to nodes based on them.

5 Conclusions and Future Work

We have shown a solution for execution of scientific workflows in hybrid clouds with cloud bursting and transparent data access. Our solution enables portable deployment of scientific workflows into two or more clouds as long as they support Kubernetes clusters, as most large public cloud providers currently do. Transparent data access hides data transfer between clouds and allows for block-level optimization wherein only the required file blocks are transferred. We have used graph partitioning to ensure load balancing of the workload across two clouds while minimizing the communication volume. These capabilities were evaluated experimentally using the HyperFlow workflow management system integrated with the Onedata data management platform and deployed in a hybrid cloud comprising two Kubernetes clusters located in the Cyfronet AGH's private cloud and the Google Cloud. Future work involves more detailed analysis and experiments to investigate the impact of several factors on the execution performance and quality of workflow partitioning, including the accuracy of job resource requests, and accurate graph vertex and edge weights, reflecting data access patterns in scientific workflows.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

Acknowledgements. The research presented in this paper has been partially supported by the funds of Polish Ministry of Science and Higher Education assigned to the AGH University of Science and Technology.

References

1. Afgan, E., Coraor, N., Chilton, J., Baker, D., Taylor, J., Team, G.: Enabling cloud bursting for life sciences within galaxy. Concurrency and Computation: Practice and Experience 27(16), 4330–4343 (2015)
2. Balis, B.: Hyperflow: A model of computation, programming approach and enactment engine for complex distributed workflows. Future Generation Computer Systems 55, 147 – 162 (2016)
3. Balis, B., Figiela, K., Jopek, K., Malawski, M., Pawlik, M.: Porting HPC applications to the cloud: A multi-frontal solver case study. Journal of Computational Science 18, 106–116 (2017)
4. Belgacem, M.B., Chopard, B.: A hybrid hpc/cloud distributed infrastructure: Coupling ec2 cloud resources with hpc clusters to run large tightly coupled multiscale applications. Future Generation Computer Systems 42, 11–21 (2015)
5. Bicer, T., Chiu, D., Agrawal, G.: A framework for data-intensive computing with cloud bursting. In: 2011 IEEE international conference on cluster computing. pp. 169–177. IEEE (2011)
6. Chang, Y.S., Fan, C.T., Sheu, R.K., Jhu, S.R., Yuan, S.M.: An agent-based workflow scheduling mechanism with deadline constraint on hybrid cloud environment. International Journal of Communication Systems 31(1), e3401 (2018)
7. Da Silva, R.F., Chen, W., Juve, G., Vahi, K., Deelman, E.: Community resources for enabling research in distributed scientific workflows. In: 2014 IEEE 10th International Conference on e-Science. vol. 1, pp. 177–184. IEEE (2014)
8. Łukasz Dutka, Wrzeszcz, M., Lichoń, T., Słota, R., Zemek, K., Trzepla, K., Łukasz Opioła, Słota, R., Kitowski, J.: Onedata — a step forward towards globalization of data access for computing infrastructures. Procedia Computer Science 51, 2843 – 2847 (2015), international Conference On Computational Science, ICCS 2015
9. Goonasekera, N., Mahmoud, A., Chilton, J., Afgan, E.: Galaxycloudrunner: enhancing scalable computing for galaxy. BioRxiv (2020)
10. Guo, T., Sharma, U., Shenoy, P., Wood, T., Sahu, S.: Cost-aware cloud bursting for enterprise applications. ACM Transactions on Internet Technology (TOIT) 13(3), 1–24 (2014)
11. Hazekamp, N., Kremer-Herman, N., Tovar, B., Meng, H., Choudhury, O., Emrich, S., Thain, D.: Combining static and dynamic storage management for data intensive scientific workflows. IEEE Transactions on Parallel and Distributed Systems 29(2), 338–350 (2017)
12. Ilyushkin, A., Ghit, B., Epema, D.: Scheduling workloads of workflows with unknown task runtimes. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 606–616. IEEE (2015)
13. Lin, B., Guo, W., Lin, X.: Online optimization scheduling for scientific workflows with deadline constraint on hybrid clouds. Concurrency and Computation: Practice and Experience 28(11), 3079–3095 (2016)
14. Liu, Y., Khan, S.M., Wang, J., Rynge, M., Zhang, Y., Zeng, S., Chen, S., dos Santos, J.V.M., Valliyodan, B., Calyam, P.P., et al.: Pgen: large-scale genomic variations analysis workflow and browser in soykb. In: BMC bioinformatics. vol. 17, p. 337. BioMed Central (2016)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21

15. Liu, Z., Xiang, T., Lin, B., Ye, X., Wang, H., Zhang, Y., Chen, X.: A data placement strategy for scientific workflow in hybrid cloud. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). pp. 556–563. IEEE (2018)
16. Marathe, A., Harris, R., Lowenthal, D.K., De Supinski, B.R., Rountree, B., Schulz, M., Yuan, X.: A comparative study of high-performance computing on the cloud. In: Proceedings of the 22nd international symposium on High-performance parallel and distributed computing. pp. 239–250 (2013)
17. Mell, P., Grance, T., et al.: The nist definition of cloud computing (2011)
18. Moulitsas, I., Karypis, G.: Architecture aware partitioning algorithms. In: International Conference on Algorithms and Architectures for Parallel Processing. pp. 42–53. Springer (2008)
19. Netto, M.A., Calheiros, R.N., Rodrigues, E.R., Cunha, R.L., Buyya, R.: Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges. ACM Computing Surveys (CSUR) 51(1), 1–29 (2018)
20. Orzechowski, M., Balis, B., Pawlik, K., Pawlik, M., Malawski, M.: Transparent deployment of scientific workflows across clouds-kubernetes approach. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). pp. 9–10. IEEE (2018)
21. Parashar, M., AbdelBaky, M., Rodero, I., Devarakonda, A.: Cloud paradigms and practices for computational and data-enabled science and engineering. Computing in Science & Engineering 15(4), 10–18 (2013)
22. Tanaka, M., Tatebe, O.: Workflow scheduling to minimize data movement using multi-constraint graph partitioning. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). pp. 65–72. IEEE (2012)
23. Tchernykh, A., Schwiegelsohn, U., Alexandrov, V., Talbi, E.g.: Towards understanding uncertainty in cloud computing resource provisioning. Procedia Computer Science 51, 1772–1781 (2015)
24. Wu, H., Ren, S., Garzoglio, G., Timm, S., Bernabeu, G., Kimy, H.W., Chadwick, K., Jang, H., Noh, S.Y.: Automatic cloud bursting under fermicloud. In: 2013 International Conference on Parallel and Distributed Systems. pp. 681–686. IEEE (2013)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_21