Migrating Scientific Workflows to the Cloud

Through Graph-partitioning, Scheduling and Peer-to-Peer Data Sharing

Satish Narayana Srirama, Jaagup Viil
Mobile & Cloud Lab, Institute of Computer Science
University of Tartu
J. Liivi 2, Tartu, Estonia
Email: srirama@ut.ee

***Abstract*—In recent years cloud computing has raised significant interest in the scientific community. Running scientific experiments in the cloud has its advantages such as elasticity, scalability and software maintenance. However, the communication latencies added by the virtualization, the technology behind the cloud's service provisioning in general, is observed to be the major hindrance for migrating scientific computing applications to the cloud. The problem escalates further when we consider scientific workflows, where significant data is exchanged across different tasks. So to migrate scientific workflows to the cloud, we propose a way to reduce the data communication by partitioning and scheduling the workflow and adapting a peer-to-peer data sharing among the nodes. Different size Montage workflows were considered for the analysis of the approach. From the study, we observed that the partitioning along with the peer-to-peer file sharing reduced the data communication in cloud up to 80%.**

Keywords*—*Scientific workflows, graph partitioning, cloud, Montage, METIS, peer-to-peer.

I. Introduction

Scientific computing is a field of study that applies computer science to solve scientific problems from domains such as astronomy, genomics, material science, bioinformatics, computational chemistry etc. It is usually associated with large scale computer modeling and simulation and it often requires large amounts of computer resources. Cloud computing suits well in solving these high performance computing (HPC) problems, with its promise of provisioning virtually infinite resources. For that reason, a lot of people, especially scientists, are trying to migrate their applications, in most cases, scientific workflows, to the cloud.

Significant research [1], [2], [3] has been performed in migrating scientific computing applications to the cloud. A number of science clouds [4], [5], [6] have been established in recent years, in which scientists have run many simulations and applications and measured their performance to evaluate the applicability of doing science on the cloud. However, the idea of running scientific computing applications on the cloud has not been well received by all quarters of the scientific computing community, as the performance of cloud still lags behind when compared to grids or using computer clusters directly. The communication latencies added by the virtualization technology is observed to be the major hindrance for executing scientific computing applications on the cloud [1], [7]. The communication latencies not only increase the execution times but also raise the utility computing costs.

To counter the problem with additional communication latencies, we propose remodeling and scheduling the scientific applications (also applicable for enterprise applications), especially workflows, in a way that increases the intra-instance communication while reducing inter-instance communication, so that the applications will fit nicely to the cloud. This paper specifically targets at our study with workflow scheduling to reduce data transmission during the workflow execution by exploiting graph partitioning approaches. Partitioning helps to find the most beneficial way to run the workflow components in such a way, that the communication between the instances is reduced. Graph partitioning has been studied extensively and there are several algorithms one can consider [8]. As part of the study we have adapted multilevel k-way partitioning algorithm [9].

Scientific workflows such as Montage [10], CyberShake [11] or SIPHT [12] can benefit from such a study. For the analysis provided in this paper, we considered Montage [10], an astronomy application that was created by the NASA/IPAC Infrared Science Archive. We used graph partitioning library METIS [8], Pegasus with Condor for workflow execution and peer-to-peer file management with Mule on Pegasus. A detailed analysis is also provided quantitatively showing the gain in data transmission reduction. Rest of the paper is organized as follows.

Section II, discusses the limitations of scientific computing on the cloud. Section III later introduces Montage scientific workflow. Section IV discusses in detail, the approach of partitioning and scheduling scientific workflows for cloud migration along with detailed performance analysis. Section V provides the related work, while section VI concludes the paper with future research directions.

II. Limitations of Scientific Computing on the Cloud

Scientific computing is usually associated with large scale computer modeling and simulation and thus requires large amounts of computer resources. Traditionally, once the scientific computing applications are developed, they are executed on the Grid infrastructures, clusters and super computers. While these infrastructures are highly efficient in performing compute-intensive parallel scientific computing applications, they provide little control to the user in general and are heavily biased by the availability of the computational resources. The introduction of commercial cloud infrastructures such as Amazon EC2 and GoGrid allowed users to have access to

computer clusters fairly easily and shifted the focus from traditional approach to cost-to-value of the experiments.

Moreover, with the availability of open source cloud infrastructure software such as Eucalyptus, Open Nebula and OpenStack, it has become simple to establish private clouds and to test the applications well before migrating them to the public clouds. Private cloud lets us first to investigate the limitations of the clouds and tweak the applications before deploying them on public infrastructure. We have created our own private cloud, SciCloud [5], established with Eucalyptus technology on XEN hypervisor, to study this approach. SciCloud has a maximum capacity of around 350 processor cores at its disposal. Recently we also have joined an OpenStack based private cloud to the SciCloud, which is basically used for the analysis provided in this paper. The possibility of dynamically adding more instances from the public clouds when the need arises makes the use of private clouds more interesting. SciCloud has customised machine images with support for several typical scientific computing and simulation tools such as Python with NumPy, SciPy, and Scilab. Moreover, scripts have been developed, which prepare the complete deployment for MPI (Message Passing Interface - used for creating parallel applications) applications on SciCloud.

We have performed detailed analysis with several benchmark applications such as NASA Advanced Supercomputing Parallel Benchmarks (NAS PB) and custom matrix-vector multiplication implementations to observe the execution of scientific computing applications on different cloud platforms. We compared the results with experiments performed without the cloud platform. The idea was to identify and measure the true latency costs added by moving applications to the cloud. Both the cloud and cluster experiments ran on the same physical hardware. We have run CG and EP (Embarrassingly Parallel) problems from NAS PB. CG is an iterative algorithm for solving systems of linear equations. The general idea of CG is to perform an initial inaccurate guess of the solution and then improve the accuracy of the guess at each following iteration using different matrix and vector operations. In EP benchmark, two-dimensional statistics are accumulated from a large number of Gaussian pseudo-random numbers, which are generated according to a particular scheme that is well-suited for parallel computation. This problem is typical of many Monte Carlo applications [13]. The former is used to evaluate transmission delays: latency and bandwidth, while the latter is used for testing processor performance.

processors	1	2	4	8	16
CG on cluster	167.5	83	65	55	31
CG on cloud	161.1	82.4	68.1	74	156.7
EP on cluster	143.4	86.1	42.2	20.3	
EP on cloud	130.5	65.1	33.6	16.9	

TABLE I: Execution times (sec) of CG and EP from NAS PB

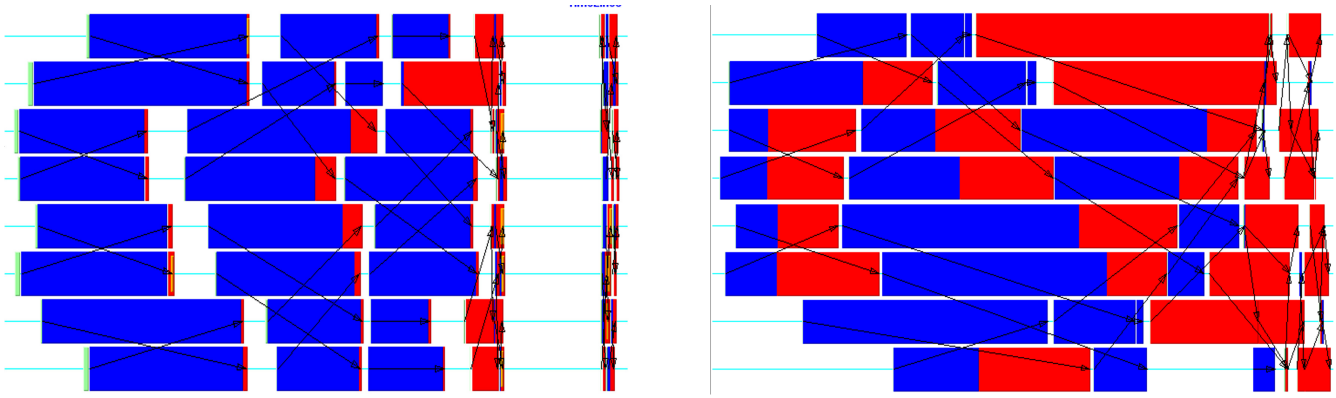
Table I summarizes the results. The performance gain achieved by adding more machines to the setup is comparable in both the cloud and cluster cases. However, the CG times rise significantly with the number of processors on the cloud, starting at 8 instances. The EP times did not show any slowdown on the cloud. To uncover the reasons for such high run times the experiments were run with MPE Profiler that shows MPI function calls and hence allows distinguishing calculations from communication. The NAS CG problem of Class B runs CG algorithm multiple times and each algorithm makes 75 iterations. Each iteration consists of two parts, calculation and communication. The times spent in `MPI_Send` (darker/blue) and `MPI_Wait` (lighter/red) are shown on Figure 1. The time spent on the calculation part in one CG iteration is approximately 26 and 24 milliseconds for cluster and cloud respectively. The time spent on the communication part are very different: 2.5 and 13 milliseconds. This makes CG time for 8 processes on the cloud about 50% slower. The reason for such high transmission latency in the cloud case is due to the virtualization technology. Moreover, we also have observed that MPI applications are very sensitive to the number of VMs per computer. Also the latencies and processing capabilities are much worse when multiple virtual machines are allocated per one machine core. This leaves a lot of research scope at the virtualization technology to address these problems [1].

III. Montage and Scientific Workflows

Workflows have lately become a standard for managing and representing complicated scientific computations [14]. It provides an abstracted view over the experiment that is being performed [15]. Each computation may contain thousands of tasks that are executed, in an order, on top of programs such as Pegasus or Kepler. One can say that a scientific workflow is the method of bringing data and processes together into a structured set of steps to overcome a scientific problem. The ability to map complex computations to a series of tasks allows the workflows to be runnable on grids and clouds and therefore, scientist are not held back by computational resources any more.

Every workflow has its unique arrangement, but it usually consists one of the five basic workflow structures: process, pipeline, data distribution, data aggregation or data redistribution [16]. Process is one of the simplest structures, which takes some input to produce an output. Pipeline consists of several processes and is the most common part in workflows. Data distribution takes some input and outputs multiple data that is consumed by various of tasks. In some cases data distribution is used to divide a large dataset to smaller subsets for easier processing for the next tasks. Data aggregation acquires for input, multiple outputs from other jobs, and produces a combined data product. Data redistribution is the combination of data aggregation and distribution, which takes multiple inputs and outputs numerous datasets.

There are several scientific workflows which are popular among the respective communities such as CyberShake, a seismic hazard analyzer, SIPHT a bioinformatic workflow dealing with the search of untranslated RNAs, Epigenomics [17], a genome sequencing workflow and etc. Of these, Montage is a popular astronomy application that was created by the NASA/IPAC Infrared Science Archive. The open source program is made for generating custom mosaics of the sky using images in the Flexible Image Transport System (FITS) format. Montage graphs can vary in size, for example, a 1.0 degree square mosaic contains of approximately 387 tasks and 84 input images. Figure 2 shows a small, twenty node montage workflow along with a note of the tasks being performed across the layers.



(a) cluster: 2.5ms (b) cloud: 13.0ms

Fig. 1: Communication pattern of a CG iteration with 8 nodes on the cluster and cloud. The time intervals of the iterations are shown under the respective profiling diagrams.

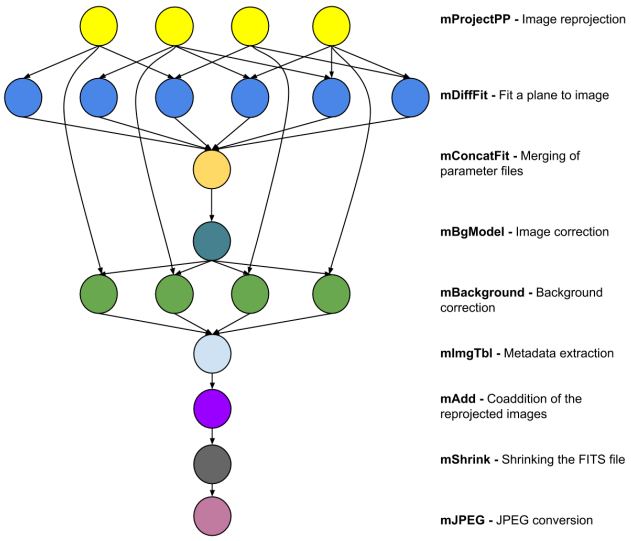


Fig. 2: Montage Workflow

The first task of the workflow is mProjectPP. The number of these jobs is dependent on the number of FITS images (each one is ∼2.1 MB) provided. The outputs, each approximately ∼8.4 MB of size, are re-projected images and area images that are a fraction of the image that is in the final mosaic. Next job mDiffFit calculates the difference for each pair of overlapping images and outputs the data (less than a KB ∼250 bytes) to mConcatFit. After fitting the images, the job outputs the data of a size ∼2.9 KB to mBgModel, which applies a correction to each image for a better global fit and sends ∼470 bytes of data to the next job. The next job is mBackground, where background correction is applied to each image. After that, the mImgTbl, which receives ∼4.2 MB of data from each mBackground job, extracts the metadata from the images and creates an image metadata table which is used by several other programs. The mAdd takes for input approximately ∼8.4 MB of data. It is the most computation heavy job which co adds the re-projected images to form the final mosaic in the FITS format and outputs also an area image, ∼8.4 MB of size. The FITS file is later reduced by the mShrink job, which outputs the shrunken file (∼4.2 MB) to mJPEG that converts the image to JPEG format. Based on these input and output sizes, we can construct a weighted graph of the Montage workflow. The graph is used in the partitioning and scheduling the workflow, which is discussed further in the next subsections.

From the Montage scenario, we can see that in scientific workflows, a lot of data is exchanged across several jobs and if the workflow is migrated to and executed in an environment which is not ideal for data transmission, such as cloud, it increases the execution times as well as the utility computing costs. However, cloud with its well known features like elasticity and on-demand resource provisioning, makes it ideal for migrating the workflows and also gives the scientists a new dimension - cost to value of the experiments. So, to fit the scientific workflows for the cloud environment and at the same time deal with the transmission latencies problem, it is a good idea to schedule the job execution in such a way that increases the intra-instance communication while reducing the inter-instance communication.

IV. Partitioning and Scheduling Scientific Workflows

In general any workflow (scientific or enterprise) can be represented as a directed acyclic graph (DAG), $G = (V, E)$, where $V = v_1, v_2, \ldots, v_n$ are the tasks (units of computation) performed in the workflow and edges $E \subset V \times V$. Edges encode data dependencies and if $(v_i, v_j) \in E$, tasks v_i and v_j are neighbors and v_j is directly dependent on the output of v_i. The weight of the edge, $W_{i,j}$ denotes the amount of data transferred from v_i to v_j. The workflow can be executed on a cluster or a set of instances rented from the cloud based on the utility computing model. To reduce the inter-instance communication in the deployment cluster, we have to partition the workflow, so that the sum of the weights of the edges connecting to vertices in different groups is minimized. After partitioning the workflow-graph to approximately equal sized parts, we can assign the partitioned subsets of the workflow to run on the cluster.

A. Partitioning the workflow

The partitioning problem is defined as follows: Partition the vertices of the graph into k approximately equal partitions such that the sum of the weights of edges that are connected to vertices in different parts is minimized. The k-way partitioning problem can be formulated to: Given a graph $G = (V, E)$ with $|V| = n$, partition V into k subsets, $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \phi$ for $i \neq j, |V_i| = n/k$, and $\cup_i V_i = V$, and the sum of the weights of edges of E whose incident vertices belong to different subsets is minimized. The partitioning of V is generally represented by a partitioning vector P, with a length n. So for every node $v \in V$, $P[v]$ is an integer between 1 and k that shows to which partition vertex v belongs.

It is known that graph partitioning problem is NP-complete. Graphs and graph partitioning are used in numerous areas of computing such as social networks, scientific and distributed computing, biological networks, etc [18]. METIS is a set of serial programs for partitioning graphs and the algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes [8]. As part of the study we used multilevel k-way partitioning [9] for creating homogenous groups, which is the default and probably the best partitioning algorithm as per METIS. Moreover, METIS's multilevel k-way partitioning algorithm provides additional capabilities, when compared to multilevel recursive bisection [19], such as minimize the resulting subdomain connectivity graph, enforce contiguous partitions, minimize alternative objectives, etc.

The basic idea behind the multilevel k-way partitioning algorithm can be summarized as follows. First the graph $G = (V, E)$ is coarsened down to a small number of vertices (coarsening phase). During the coarsening phase a progression of smaller graphs is created from the original graph. Note that each coarser graph has both the connectivity and the ratio of the initial graph. After coarsening, the next stage is the partitioning phase, where a k-way partitioning of the coarsest graph is calculated, which is computed using a multilevel bisection algorithm. After that the coarsest graph is partitioned (initial partitioning phase) the result is projected back towards the original graph (uncoarsening phase). To improve the final partitioning, it is refined during the projection back to the original graph [9].

B. Scheduling the workflow

Once the workflow is partitioned, we used Pegasus framework, for scheduling and submitting jobs to run on top of multiple instances on cloud. Pegasus is a popular framework for mapping complex scientific workflows onto distributed systems [20]. Pegasus has numerous features such as portability, reliability, scalability, etc. With Pegasus, workflows can easily be run in different environments such as grids, clouds, campus clusters and so on (portable), jobs that fail are rescheduled to run again (reliable), and Pegasus can run different size workflows (scalable) on top of different type of resources.

The main components of Pegasus are:

- *Mapper (Pegasus Mapper)*: Makes a runnable workflow from an abstract workflow. It also searches for the software and computational resources where the workflow should be executed.
- *Execution Engine (DAGMan)*: Executes the tasks that are defined in the workflow.
- *Task Manager (Condor Schedd)*: Manages the tasks and their execution.
- *Monitoring Component (Pegasus Monitord)*: The component that monitors all the processes, creates the logs etc.

Pegasus in turn utilizes Condor, a specialized workload management system for compute-intensive jobs [21]. Users just have to submit the job to Condor and the rest of the tasks such as job queuing, prioritizing, monitoring, etc. are all taken care by Condor. Condor consists of the following main daemons: condor_master, *condor_startd*, *condor_starter*, *condor_collector*, *condor_negotiator*, *condor_schedd* and *condor_shadow*. The *condor_master* is the daemon that is responsible for starting and managing other daemons and it runs on every machine in the Condor pool. A condor pool is the cluster of machines where the jobs are run. The *condor_startd* is the daemon that advertises information about the current machine and enables the machine to run or stop jobs in the Condor pool. *Condor_starter* is activated by *condor_startd* and it handles the managing and starting of a job. *Condor_collector* is the daemon that collects all the information such as updates sent by other daemons (ClassAd) to this daemon. The *condor_negotiator* daemon matches the jobs to the machines from the information it gets from the collector and *condor_schedd* submits the jobs to the queue. The *condor_shadow* runs on every machine, where the job is executing and it is responsible for logging and requesting for file transfers when the job has completed.

To submit a workflow to Pegasus one has to provide the following files:

- *Directed Acyclic Graph in XML (DAX) file* - This is the file that contains the description of the jobs and their dependencies.
- *Transformation catalog file* - Describes all the executables that the workflow needs to run the jobs.
- *Replica catalog file* - A file that contains mappings to the files that the workflow needs.
- *Sites catalog file* - This file describes all the sites where the workflow tasks are going to be executed.

C. Montage and Pegasus on cloud

To observe the migration of scientific workflows to the cloud, we partitioned the Montage workflow, 0.2 degree square mosaic, with 8 input images, into 3 groups. We later executed the workflow on three cloud instances using Pegasus. Figure 3 shows the total amount of data that is transmitted across the different nodes during the complete workflow execution, with and without the partitioning and scheduling, across three different runs. From the analysis, we observed that the reduction in the data transmission is not so significant even after partitioning and the data transmission have always varied in quantity (as shown in figure 3).

A thorough analysis revealed that one of the bottlenecks of running Pegasus with Condor in the cloud is the file sharing

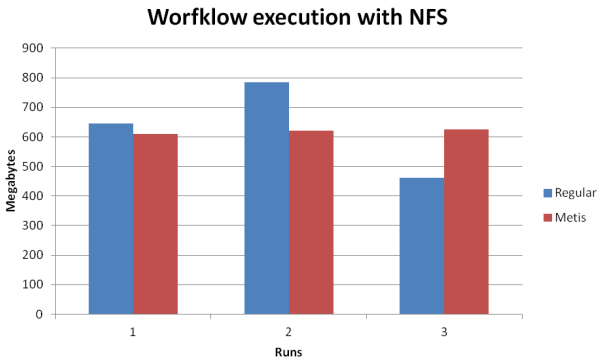


Fig. 3: Data transfer comparisons between random scheduling and partitioning with METIS. The values are shown for 3 random runs where the gain with partitioning is ambiguous

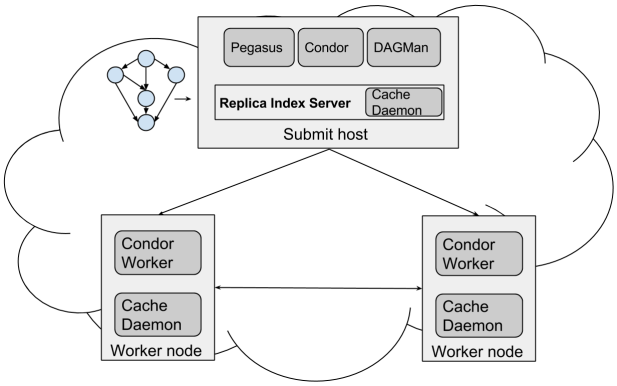


Fig. 4: The final setup of the workflow execution in the cloud (3 instances) with peer-to-peer file sharing

- all the communication goes through the central manager. This means to execute a task on a node the input is first copied from the centralized file system or a Network File System (NFS) [22] and the output is copied back to the NFS once the execution is finished, via the central manager. Workflow partitioning in this scenario will not help to reduce the communication between the instances, because it does not matter anymore where the jobs are being run, every data exchange goes always through the central node. This type of centralized approach is lot more logical in grid and supercomputing environments.

Scientific workflows are traditionally designed for grids and clusters, where instances are dynamically allocated and are revoked once the subtasks are completed. So no information is retained on the nodes and the results are stored in the centralized file system. However, when executing the workflow on dedicated infrastructure as in the case of cloud, it is a good idea to store the intermediary results of the workflow task executions on the nodes, so that the next tasks do not have to download the data again from the central repository. This would significantly reduce the amount of data that is exchanged across different nodes, which is crucial for the cloud migration.

D. Pegasus execution in peer-to-peer (P2P) manner

To solve this problem, the peer-to-peer file manager (Mule) for Pegasus [23] was considered. The file manager has three components: *replica index service*, a *cache daemon*, and a *client*. Instead of storing the files on the central manager, the peer-to-peer file manager allows storing all the necessary files on each compute node. On each node where a computation occurs, the cache daemon stores the copies of the files that the job uses or outputs. To know where the files are located the replica index server is used, where all the locations of the files are listed. Replica catalog has both logical file name and the URL of a file. So if a machine needs some data, it first checks the replica catalogs list and then retrieves it. Figure 4 displays the final setup - Pegasus with Condor integrated with P2P sharing.

With the introduction of Mule for Pegasus, the complete migration procedure is shown in figure 5. We have developed an API for the transformations and several scripts which automatize the complete procedure.

First the workflow file is transformed to a file format that is accepted by METIS (① in figure 5), i.e., a text file, where each line contains the weights and connections of a single node. After converting the workflow to a text file, it is partitioned with the METIS toolkit (② in figure 5). The output of the partitioning is again a text file, where on each line is an integer showing to which group a node belongs to. The output file is transformed again to an abstract workflow file (③ in figure 5), where under each job, the machine where it is going to run is specified (executionPool), as specified below.

```
<profile namespace="hints"
key="executionPool">mainnode</profile>
```

After the creation of the abstract workflow, where each job is assigned to a compute node (scheduled), the workflow is executed on the cloud (④ in figure 5). For example, for planning a workflow to run on top of three instances, we used the following command:

```
pegasus-plan \
    --conf pegasus.properties \
    --sites mainnode, worker1, worker2 \
    --output-site local \
    --staging-site mainnode \
    --dir submit \
    --dax m3_dag.xml \
    --nocleanup \
    --submit\
```

The experiments were later performed again, with 0.2, 1.0 and 2.0 degree Montage workflows, this time introducing the Mule. With the developed API and scripts we can partition any custom size Montage workflow and run it in the cloud with ease. During the analysis we observed that only with the implementation of the P2P file sharing on top of Pegasus, the overall data communication in the cluster was already reduced over 50%, even without introducing the partitioning. For example, running a 1.0 degree Montage workflow without P2P on three compute nodes results approximately 4 gigabytes of data exchange and with the P2P implementation only 2 gigabytes was exchanged.

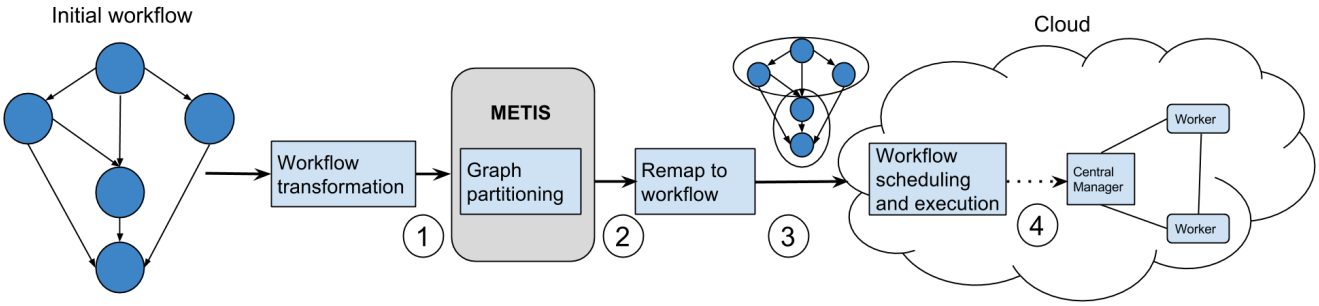


Fig. 5: The complete scientific workflow migration procedure

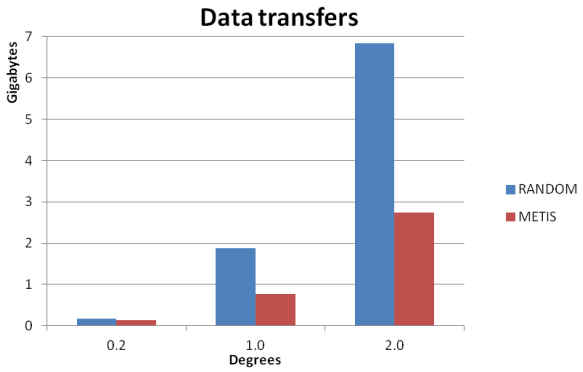


Fig. 6: Data transfer comparisons between random scheduling and partitioning with METIS. The values are shown for Montage 0.2, 1.0 and 2.0 degree problems.

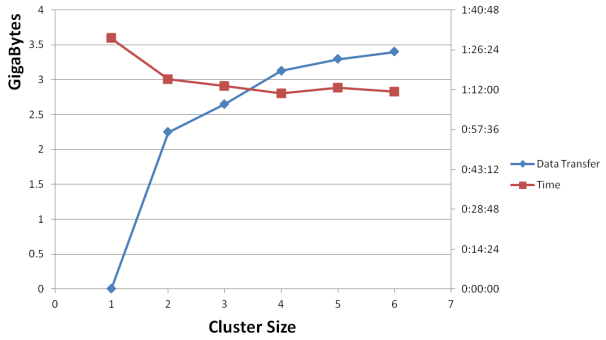


Fig. 7: Workflow execution across varying cluster sizes

From the analysis we could also observe that, the overall communication between the instances was reduced up to 60% with the help of partitioning the workflow with METIS. With the P2P approach and the partitioning the overall data communication was reduced up to 80%. For example, the data transfer on the 2.0 degree workflow case was reduced from ∼15.5 GB to ∼7.5 GB with the help of Mule, which further got reduced to ∼2.8 GB with both partitioning and Mule. Figure 6 displays the data communication of different size Montage workflows.

E. Finding ideal number of partitions

While the results show that there is significant reduction in transmission latencies while executing the workflow, one problem still remains is how to decide the ideal number of groups into which the workflow can be partitioned. For any parallelizable application, there is only to a certain level the parallelization is cost effective, beyond which the parallelization overhead itself will eclipse the achieved gain. This is an analysis which is application specific and also depends on input and problem size, type of instances etc. However, the problem size of scientific workflows can be generalized in certain cases.

To be more specific, if we consider the Montage workflow, we can consider different input sizes for the problem. For example, as already mentioned, 0.2, 1.0 and 2.0 degree square mosaic will always have approximately 8, 84 and 312 input images, respectively. This fixes the number of tasks performed during the workflow execution and the amount of data being exchanged. So, if we can find the ideal number of partitions for one particular size of problem, the amount is applicable to any other input with similar problem sizes.

To decide the ideal number of partitions, we considered a particular problem size (2.0 degree Montage is discussed here) and tried to partition the workflow into different sizes and measured the execution times and the data transmission latencies. Figure 7 shows the workflow execution across varying cluster sizes. From the figure we can observe that partitioning to 4 groups is ideal, in terms of execution time, achieved parallelization and incurred utility computing costs, for that particular problem. So for every input of the same size the same cluster size could be utilized. However, this analysis is exhaustive enough and is applicable only when it is common to consider particular problem sizes, which is generally observed in scientific workflows.

Once the ideal cluster size has been identified, we migrated the workflow to the Amazon cloud (on 4 *m1.small* instances) and executed it for the 2.0 degree Montage. Figure 8 shows the data communication latencies, for both the cases where workflow is scheduled randomly and where workflow is adapted for cloud migration by partitioning and scheduling. The data communication in the Amazon cluster with the 2.0 degree Montage workflow was reduced after partitioning nearly by ∼70%.

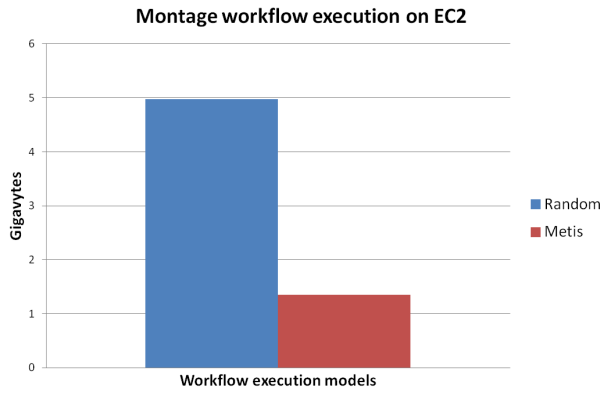


Fig. 8: Data communication across 4 nodes in Amazon EC2 while performing 2.0 degree Montage workflow execution

V. Related work

Alexandru Iosup et al. [24] studied the performance of cloud services for scientific computing tasks. They state that cloud holds great promise for the scientific computing community as it can be a cheap alternative to supercomputers and specialized clusters, more reliable platform than grids, and more scalable platform than the largest commodity clusters. Their motivation was that the potential of clouds for scientific computing has been largely unexplored. They evaluate the performance of four commercial cloud platforms, Amazon EC2, GoGrid, Elastic Hosts and Mosso and compare their performance for Many-Task Computing based scientific computing workloads. They conclude that current clouds need an order of magnitude in performance improvement to be useful to the scientific community and propose improvements for supporting scientific computing workloads in the cloud.

As previously mentioned, significant research [25], [24], [26] has been performed in migrating scientific computing applications to the cloud. Most of these studies, similar to our results [1], iterate that virtualization is the major hindrance for migrating scientific computing or, in general, high performance computing applications to the cloud. Some of these studies [24] are quite extensive and are more detailed when compared to our work. However, when SciCloud [5] was established in 2010, this was one of the first questions we asked ourselves: what are the major hindrances for migrating scientific computing applications to the cloud? With the discussed experiments, and our access to private cloud infrastructure, we could clearly measure the latencies added by virtualization. For example, to see the clear network latencies added by the virtualization, we restricted the cloud instances to start on single physical node in a cluster. This sort of deployment guarantees are not offered by public clouds.

In the area of workflow partitioning Çatalyürek et al. used a heuristic called DPTA (data placement and task assignment tool) to optimize the execution of scientific workflows in the cloud [27]. They achieved up to 38% of reduction in communication costs. They enhanced a multilevel hyper-graph partitioning tool called PaToH [28]. However, in our analysis we considered METIS for partitioning. Correspondingly, Chen and Deelman developed a system on top of the Pegasus to estimate, partition and schedule workflows [29] onto execution sites with storage constraints, to improve the overall runtime. They partitioned the workflow into sub-workflows as it reduced the complexity of workflow mapping. For example, an entire CyberShake workflow has 1.9×10^5 tasks. In contrast our work schedules separate workflow tasks to the running sites individually, because Montage workflow cases used for the tests, does not contain so many tasks. Similarly, M. Tanaka and O. Tatebe used METIS to minimize the data movement between the nodes [30]. In their case they used a Pwrake parallel workflow system, whereas we considered workflow execution using Pegasus toolkit with P2P implementation.

Regarding, running scientific workflows in the cloud and the challenges of it; Juve and Deelman studied the problem and state that the benefits of running scientific workflows are for example lease based provisioning, elasticity and the support for legacy applications [31]. They too agree that a lot of work is needed to be done to bring the performance up to the level of grids. For this analysis, they have compared different environments to run scientific workflows, e.g., EC2 [32] and Open Science Grid [33]. According to them a lot of effort has gone into improving running workflows in the cloud - many scientists have been developing multiple algorithms to take advantage of the pricing model and elasticity of infrastructure clouds. One can say that nowadays there is not anymore a drastic difference between running workflows on the grid or on the cloud, both have their advantages and disadvantages. However, the most common challenges or disadvantages of running workflows in the cloud are for example system administration, complexity, data management and cost.

VI. Conclusions and Future Work

Migrating scientific applications/workflows to the cloud is interesting in terms of achieved advantages such as elasticity, scalability, utility computing backed new perception of cost-to-value of experiments etc. However, cloud also has certain limitations, which actually confines its performance for scientific/HPC calculations, such as virtualization and the communication latencies added by it. So migrating scientific application to the cloud needs a fresh perspective, which can take benefit from cloud's advantages and yet can counter its limitations. This leaves significant scope for new frameworks, tools and methodologies specifically targeting at cloud migration.

In this paper, we have proposed a way for reducing the data communication of scientific workflows between the computation nodes in the cloud. The communication reduction is achieved with workflow partitioning using the METIS toolkit and for the partitioning we used the multilevel k-way algorithm. We observed that the data transfer between the nodes was not reduced after partitioning, in the centralized file system case, where all the data goes through one node, which is prominent in scientific experiments performed on traditional infrastructure such as clusters, grids and supercomputers. However, to overcome this problem, especially with cloud migration being the final goal, we adapted the peer-to-peer data sharing principle on top of Pegasus. For that we used Mule, which allows all the compute nodes to communicate with each other.

With the adaptation of P2P sharing alone the data transfer on the cluster was reduced over 50%. After the partitioning

with METIS and rescheduling the workflows in the cloud, we observed a communication reduction up to 60%. For example in the 1.0 degree Montage workflow, the data communication was reduced from 4 gigabytes of data exchange to 2 gigabytes with the peer-to-peer model alone. After partitioning the workflow with METIS, the communication was further reduced down to 700 megabytes. So, with the P2P approach and partitioning, we managed to reduce the data communication in the cluster up to 80%.

Regarding the future work, the discussed partitioning methods result in homogeneous partitions, considering only the data transfer across the nodes. We are planning to extend the partitioning also to include the amount of processing performed on each node. This should result in homogeneous partitions both in terms of processing as well as data transfer. We are also interested in non-homogeneous partitions, where each partition can result in different size, still optimizing the communication latencies. These sorts of non-homogeneous partitions can take advantage of heterogeneity of cloud instances. However, scaling such a system would be very difficult and we are currently in the process of designing an ideal deployment configuration for such a system based on in-coming loads, using the linear programming models, especially for enterprise workflows.

ACKNOWLEDGMENT

This work is supported by European Regional Development Fund through EXCS, Estonian Science Foundation grant PUT360 and Target Funding theme SF0180008s12.

REFERENCES

[1] S. Srirama, O. Batrashev, P. Jakovits, and E. Vainikko, "Scalability of parallel scientific applications on the cloud," *Scientific Programming*, vol. 19, no. 2, pp. 91–105, 2011.

[2] C. Bunch, B. Drawert, and M. Norman, "MapScale: A Cloud Environment for Scientific Computing," University of California, Tech. Rep., 2009.

[3] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Pervasive Systems, Algorithms, and Networks (ISPAN)*, dec. 2009, pp. 4–16.

[4] Helix Nebula, "The Science Cloud," [accessed 05-May-2014]. [Online]. Available: http://www.helix-nebula.eu/

[5] S. N. Srirama, O. Batrashev, and E. Vainikko, "SciCloud: Scientific Computing on the Cloud," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010)*, 2010, p. 579.

[6] FutureGrid, "Using IaaS Clouds on FutureGrid," [accessed 05-May-2014]. [Online]. Available: https://portal.futuregrid.org/using/clouds

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[8] G. Karypis and V. Kumar, "A fast and highly quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, p. 359392, 1999.

[9] ——, "Multilevel k-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48(1), pp. 96–129, 1998.

[10] Montage, "An astronomical image engine," [accessed 05-May-2014]. [Online]. Available: http://montage.ipac.caltech.edu

[11] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner *et al.*, "Cybershake: A physics-based seismic hazard model for southern california," *Pure and Applied Geophysics*, vol. 168, no. 3-4, pp. 367–381, 2011.

[12] SIPHT, [accessed 05-May-2014]. [Online]. Available: http://newbio.cs.wisc.edu/sRNA/

[13] NASA, "NAS Parallel Benchmarks," [accessed 05-May-2014]. [Online]. Available: http://www.nas.nasa.gov/Resources/Software/npb.html

[14] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," *Ieee computer*, vol. 40, no. 12, pp. 26–34, 2007.

[15] K. Wolstencroft, P. Fisher, D. D. Roure, and C. Goble, "Scientific workflows," 2009. [Online]. Available: http://cnx.org/content/m32861/1.3/

[16] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science (WORKS 2008)*. IEEE, 2008, pp. 1–10.

[17] USC Epigenome Center, [accessed 05-May-2014]. [Online]. Available: http://epigenome.usc.edu

[18] D. LaSalle and G. Karypis, "Multi-threaded graph partitioning," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 225–236.

[19] METIS, "Manual," [accessed 05-May-2014]. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/metis/

[20] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[21] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor: a distributed job scheduler," in *Beowulf cluster computing with Linux*. MIT press, 2001, pp. 307–350.

[22] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the sun network filesystem," in *Proceedings of the Summer USENIX conference*, 1985, pp. 119–130.

[23] R. Agarwal, G. Juve, and E. Deelman, "Peer-to-peer data sharing for scientific workflows on amazon ec2," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*. IEEE, 2012, pp. 82–89.

[24] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 931–945, 2011.

[25] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 50.

[26] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in *Proceedings of the 2008 international workshop on Data-aware distributed computing*. ACM, 2008, pp. 55–64.

[27] Ü. V. Çatalyürek, K. Kaya, and B. Uçar, "Integrated data placement and task assignment for scientific workflows in clouds," in *4th int. workshop on Data-intensive distributed computing*. ACM, 2011, pp. 45–54.

[28] Ü. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1479–1487.

[29] W. Chen and E. Deelman, "Partitioning and scheduling workflows across multiple sites with storage constraints," in *Parallel Processing and Applied Mathematics*. Springer, 2012, pp. 11–20.

[30] M. Tanaka and O. Tatebe, "Workflow scheduling to minimize data movement using multi-constraint graph partitioning," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*. IEEE CS, 2012, pp. 65–72.

[31] G. Juve and E. Deelman, "Scientific workflows in the cloud," in *Grids, Clouds and Virtualization*. Springer, 2011, pp. 71–91.

[32] Amazon Inc., "Amazon elastic compute cloud (amazon ec2)," [accessed 05-May-2014]. [Online]. Available: http://aws.amazon.com/ec2/

[33] G. Juve, M. Rynge, E. Deelman, J.-S. Vockler, and G. B. Berriman, "Comparing futuregrid, amazon ec2, and open science grid for scientific workflows," *Computing in Science & Engineering*, vol. 15, no. 4, pp. 20–29, 2013.