UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Jaagup Viil

Remodelling Scientific Workflows for Cloud

Bachelor's Thesis (6 ECTS)

Supervisor: Satish Narayana Srirama , PhD

Tartu 2014

Contents

Glossary

EC2 - Amazon Elastic Compute Cloud
METIS – A set of serial programs for graph partitioning.
P2P – Peer to Peer
PaToH – Partitioning Tools for Hypergraph
SIPHT – A bioinformatic workflow dealing with the search of untranslated RNAs
GAE – Google App Engine
FITS – Flexible Image Transport System
DAX - Directed Acyclic Graph in XML
XML - Extensive Markup Language
DAGMan - Directed Acyclic Graph Manager
NFS – Network file system
URL - Uniform Resource Locator
RAM – Random Access memory
CPU – Central Processing Unit
HEM - Heavy Edge Matching
SHEM - Sorted Heavy Edge Matching
IaaS - Infrastructure as a Service
PaaS - Platform as a Service
SaaS - Software as a Service

Remodelling Scientific Workflows for Cloud

Abstract:
In recent years, cloud computing has raised significant interest in the scientific community. Running scientific experiments in the cloud has its advantages like elasticity, scalability and software maintenance. However, the communication latencies are 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.

One way to overcome this problem is to reduce the data communication by partitioning and scheduling the workflow and adapting a peer-to-peer file sharing among the nodes. Different size Montage workflows were considered for the analysis of this problem. From the study it was observed that the partitioning along with the peer-to-peer file sharing reduced the data communication in the cloud up to 80%.
Keywords:
Scientific workflows, cloud, partitioning, METIS

Teaduslike töövoogude modelleerimine pilve jaoks

Lühikokkuvõte:
Viimastel aastatel on hakanud teaduslikes kogukondades huvi pilvearvutuse vastu kasvama. Teaduskatsete läbiviimisel pilves on mitmeid eeliseid nagu elastsus, paindlikkus ja hooldatavus, kuid varasemad uuringud näitavad, et üks suurimaid probleeme teadusprogrammide jooksutamisel pilves on omavaheliste masinate andmevahetuse suurus. Üks lahendus sellele probleemile oleks tuvastada komponendid, mis omavahel palju suhtlevad ning panna nad pilves ühte kohta jooksma, et vähendada omavahelist andmevahetust. Antud bakalaureuse töös jagati (partitsioneeriti) Montage töövoo osad pilves asuvate virtuaalmasinate vahel ning rakendati valmis kirjutatud P2P süsteemi, et vähendada pilves olevat suhtlust. Tänu P2P süsteemile ja teadusprogrammi partitsioneerimisele vähendati kogu suhtlust pilves kuni 80%.
Võtmesõnad:
Teaduslikud töövood, pilvearvutus, partitsioneerimine, METIS

1 Introduction

In recent years, more and more people are starting to use cloud services because of the rising need for computing resources. Companies no longer need to spend money on system administrations, hardware repairs, power bills etc. They can simply rent the required computing resources from a provider and pay for their usage, much like electricity or water. One of the main advantages of cloud is that the resources they provide are elastic. This means that the amount of computing resources a company can provision from the cloud provider can change on demand and in real-time. Because of this, companies are not limited and restricted by computing hardware anymore.

While cloud is often used by enterprise users, the advantage it provides is also useful in the field of scientific computing. Commonly used applications in that domain are scientific workflows, which usually consist of large amount of jobs and therefore, need a lot of computing resources. Due to the increase of cloud usage popularity, a number of different problems have emerged. One for example, is how to reduce the communication latencies and load between different components talking in the cloud?

Previous research [1] [2] has shown that communication latencies are one of the major troubles with cloud migration. One way to overcome this problem is by identifying the components which communicate a lot and investigate whether it would be beneficial to locate them close together to reduce the cost of their communication.

Contribution to the goal of reducing the overall inter-instance (between instances) communication is to partition the scientific workflow into several parts. In this case, partitioning is used to divide the workflow elements between the available machines in the cloud. It helps to find the most beneficial way to run the workflow components in such a way that the communication between the instances is reduced. For that graph theory will be used. To do that one has to analyze the workflow and the data flow of the application. As it is known, a workflow can be represented as a graph. Thus in order to know the data flow of the application or a scientific workflow, one has to measure the applications communication between the graph nodes. For example if a job (1) outputs two megabytes of data to the next job (2), the weight between node 1 and 2 is respectively two megabytes. After collecting the data between the different nodes, one can create a weighted graph which is vitally important for the partitioning of the application.

For graph partitioning a tool named METIS [3] will be used. To test these ideas I have to use some kind of workflow that is easily modifiable and runnable for our purposes. Scientific workflows like Montage [4], CyberShake [5] or SIPHT [6] should be good for our case, because they are widely used and have a large scientific community behind them.

In general the main idea is to remodel how the applications nodes are placed in the deployment cluster, so one can reduce the overall communication on the cloud. To do that I will analyze the workflow, partition it and then run it on the cloud for testing.

1.1 Related work

The concept of partitioning scientific workflows to reduce communication between the components on the cloud has been widely researched. I now briefly discuss about running workflows in the cloud, the challenges of it and about the partitioning of scientific workflows.

In the area of workflow partitioning, Çatalyürek, Kaya and Uçar used a heuristic called DPTA to optimize the execution of scientific workflows in the cloud [7]. They achieved up to 38% of reduction in communications cost. They enhanced a multilevel hyper-graph partitioning tool called PaToH [8]. In this case a tool named METIS will be used. Chen and Deelman developed a system on top of the Pegasus to estimate, partition and schedule workflows [9] onto execution sites with storage constraints, to improve the overall runtime. In their case they partitioned the workflow into sub-workflows because it reduced the complexity of workflow mapping. For example, an entire CyberShake workflow has 1.9×10^5 tasks. In contrast this thesis schedules separate workflow tasks to the running sites individually, because workflow cases used for the tests does not contain so many tasks. M. Tanaka and O. Tatebe used also METIS to minimize the data movement between the computational nodes [10]. In their case they used a Pwrake parallel workflow system. In this paper for workflow execution the Pegasus toolkit with P2P implementation is used, which allows all the compute nodes to communicate with each other.

In addition, there have been several studies in investigating the execution of scientific computing workflows in the cloud. According to Juve and Deelman, the benefits of running scientific workflows are for example lease based provisioning, elasticity and the support for legacy applications [11]. They also said that a lot of work is needed to be done to bring the performance up to the level of grids. In their newer paper, also with Rynge, Vockler and Berriman [12] they compared different environments to run scientific workflows, e.g., EC2 [13] and Open Science Grid. 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. As reported by Juve et al., the most common challenges or disadvantages of running workflows in the cloud are for example system administration, complexity, data

management and cost.

1.2 Outline

Chapter 2: describes the state of the art. In this chapter there are explanations for what is cloud computing and why more and more people are starting to use the services provisioned by cloud. The basics of some popular cloud services like Amazon EC2 and Openstack [14] are also introduced – what are they and why are they used. I will also explain the meaning and the structure of scientific workflows, common platforms to run them on and so on.
Chapter 3: explains the solution proposed to solve the problem stated in this thesis. In this chapter I will discuss about the partitioning tool METIS and explain a little bit more in detail how the partitioning works.
Chapter 4: describes the contribution and the analysis of the stated problem. In this chapter there will be discussion about platforms like Pegasus and Condor – how to set them up and how to run workflows on top of them. Briefly there will be also explained how to modify Pegasus environment for P2P support. Also, the tests results and analysis are explained and discussed in this chapter.
Chapter 5: contains the conclusion about the findings and results.
Chapter 6: describes future work directions, some of which are: implementing workflow partitioning on top of enterprise service applications and others.

2 State of the Art

2.1 Science on cloud

Scientific computing is a research field that uses computer science to solve problems from material science, cosmology, genomics, computational chemistry, etc. Most of the time it is associated with large scale computer simulation and modelling and usually therefore it needs big amounts of computer resources. Cloud computing fits well in solving these particular scientific computing problems, because of the obligation of providing virtually unlimited resources.

As reported by Ian Foster et.al., the definition of cloud computing is as follows: *"A large scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet."* [15]. One can say that it is a computing style in which, commonly, resources scalable on demand are provided as a service over the Internet to users who does not need to have knowledge of the infrastructure that supports them. The provisioning of the cloud services can be divided into Infrastructure as a service (IaaS), Platform as a service (PaaS) or Software as a service (Saas) (see figure 1).

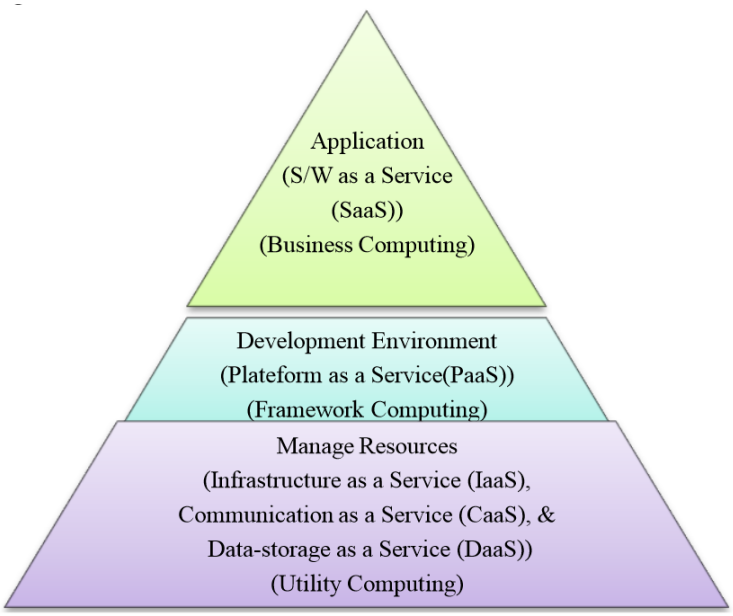

Figure 1: Basic concept of cloud computing model and services [16].

Cloud computing is basically convoluted with three of its main features: virtualization, utility computing and elasticity. A cloud platform is capable of real

time provisioning, configuring, reconfiguring, de-provisioning computing resources when it is needed. Virtualization technique forwards the idea of separating software and hardware and running multiple independent virtual servers on a single machine. Utility computing is about the idea of computing as a utility – consumers pay based on how much resources provisioned by the cloud they are using. This allows scientists to make cost projections before deployment and frees them from upfront commitment costs, i.e., pay when it is needed. Framework's elasticity can be defined as its ability to adjust correspondingly to the different amount of loads – more load means more resource provisioning and vice versa. This makes elasticity ideal for resource-intensive scientific tasks.

Traditionally when scientific computing applications are developed they are executed on the Grid infrastructures, super computers and clusters. While these environments are highly efficient in performing parallel applications that are computer-intensive, they provision limited amount of control to the user in general and are heavily dependent on the availability of computational resources. The introduction of commercial cloud infrastructures like GoGrid or Amazon EC2 granted connections to computer clusters rather easily.

Also, with the availability of open source cloud environments like Openstack, Eucalyptus or Nebula, it has become easy to set up private clouds on which to test the applications before migrating them to public clouds. Private clouds grant us the opportunity to investigate the drawbacks of the clouds and tweak the application before deploying them to public infrastructures.

While running applications on the cloud has it benefits, it has also it downsides. For example the communication and other type of latencies added by the virtualization technology is one of the major drawbacks for executing scientific computing applications on the cloud [1] [2]. Besides the performance of cloud platforms being one of the most important problems, there are problems like adapting scientific computing applications to these platforms. Most of the time cloud infrastructure is based on vast numbers of commodity computers, which are cost effective, but some amount of them is bound to fail in regular intervals. This causes problems running scientific computing applications as it needs to adjust to these failures. To deal with network or hardware failures in a distributed system, the best approach usually is to duplicate the important data and retry the computations that failed. There are frameworks that provide such fault tolerance, e.g., MapReduce [17].

Moreover, when migrating scientific computing applications to the cloud, it is usually assumed that the scientist who is performing the computation experiments has significant knowledge of computer science and cloud computing. For example scientists who are presently using grids and clusters are unfamiliar how the environments are configured, how the job queues are working and how the job is being executed. All they are interested in is that they submit a job to the queue and

can that they collect the results after some time.

As one can see, that cloud computing may have some disadvantages like communication latencies, system configurations, etc., but in the long run, qualities like elasticity, virtualization are good reasons for scientist to run their scientific computing applications on the cloud.

2.2 Cloud environments

As the popularity of cloud services is increasing the demand for cloud environments is also rising. The figure 2 illustrates the web search trends for cluster, grid and cloud computing and the spot points (A, B, .., F) indicate popular news related to cloud computing. Because of the increase of cloud computing popularity, there

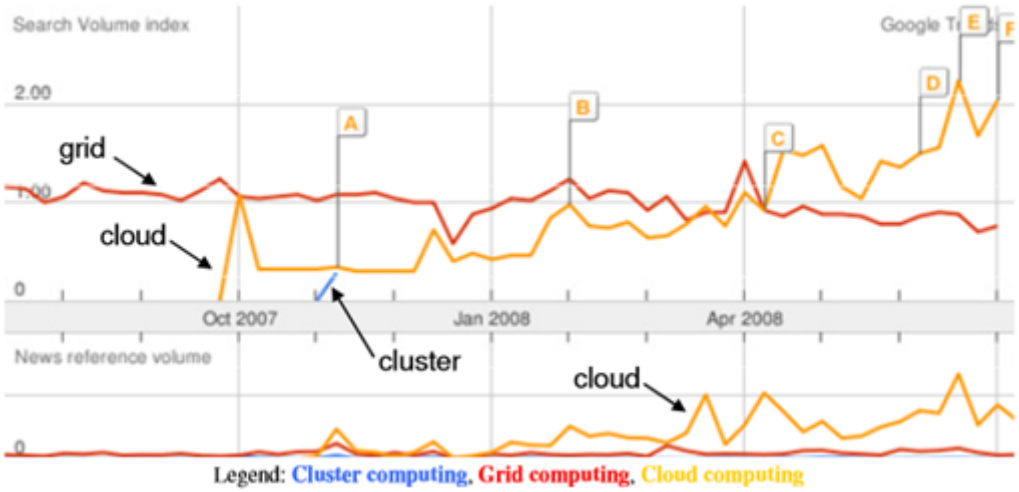

Figure 2: Google search trends for years 2007-2008 [18].

are several cloud infrastructures on the market, for example Amazon Cloud EC2, Openstack, Eucalyptus, etc. EC2 or Amazon Elastic Compute Cloud is the most well-known service of Amazon Web Services. EC2 is an infrastructure as a service (Iaas) that allows their users to rent virtual machines on which they can run their own computer applications. Users can launch, create or terminate instances when needed and they only have to pay by the hour for active servers. EC2 has many good features like elasticity, reliability, real-time monitoring, etc. For example EC2 users have access to Amazon CloudWatch that provides an interface where they can monitor CPU, network and disk metrics on their instances.

Apart from IaaS models, e.g., EC2, there are also PaaS models like Google App Engine (GAE). According to Alexander Zahariev, Google App Engine is just a platform where users can run and host their web applications on top of Google's

infrastructure [19]. Like most of the cloud service providers, Google has features like load balancing, automatic scaling, elasticity, etc. Running web applications on top of GAE is easy - a user just has to upload their product to the cloud and it is good to go.

While GAE and EC2 have many quality features, the main downside of using these environments is the cost – users have to pay for storage, instances and so forth. For these reasons there are open source projects like OpenStack or Eucalyptus that are free and provision the same features like the commercial cloud environments.

OpenStack, like EC2, is focused on IaaS. It does not differ much from the commercial clouds - users can set up websites that can scale dynamically up and down, provision more instances when needed, use it for super-computing and so on. One of the main benefits of open source clouds is that users can test their applications before running them on commercial environments – re create a live set-up before final deployment. If an application needs some changes, it is easy to modify them before migrating them to commercial clouds. On the other hand, open source cloud environments need time, effort and skills to set up. Also, one disadvantage is the lack of support – there has to be large community behind the product, i.e., forums, chat rooms, etc. do get the information that is needed. In commercial products the customer support is already included in the payment. While open source clouds are free and therefore, there are some downsides, but with proper effort it will be definitely worthwhile to set up.

2.3 Scientific workflows

Workflows have lately become a standard for managing and representing complicated scientific computations [20]. Each computation may contain thousands of tasks that are executed in an order on top of programs like Pegasus or Kepler. One can say that a scientific workflow is the process of bringing data and processes together into a structured set of steps to overcome a scientific problem.

According to Katy Wolstencroft et.al, a workflow provides an abstracted view over the experiment that is being performed [21]. It illustrates what analyses will be executed and excludes the low level details how it will be executed. It means the user does not have to understand the code behind the workflow and just the scientific protocol. The ability to map complex computation to series of tasks allows the workflows to be runnable on grids and clouds and therefore scientist are not held back any more by computational resources.

Every workflow has its each unique arrangement, but it usually consists one of five basic workflow structures: process, pipeline, data distribution, data aggregation or data redistribution [22]. Process is one of the simplest structures, which takes some input to produce an output. Pipeline consists of multiple processes

and is the most common part in workflows. Data distribution takes some input and outputs multiple data that is consumed by multiple 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 is basically the opposite of data distribution – it takes for input multiple outputs from other jobs and outputs a combined data product. Data redistribution is the combination both of data aggregation and distribution, which takes multiple inputs and outputs multiple datasets.

In this thesis a scientific workflow called Montage [4] is used. It is an 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 FITS - Flexible Image Transport System format. Montage graphs can vary in size, for example a 1.0 degree square mosaic contains of ∼387 tasks and ∼84 inputs images, while a 0.2 degree workflow has ∼35 tasks and ∼8 inputs. In figure 3 there is a small, twenty node montage workflow. Now the characterization and

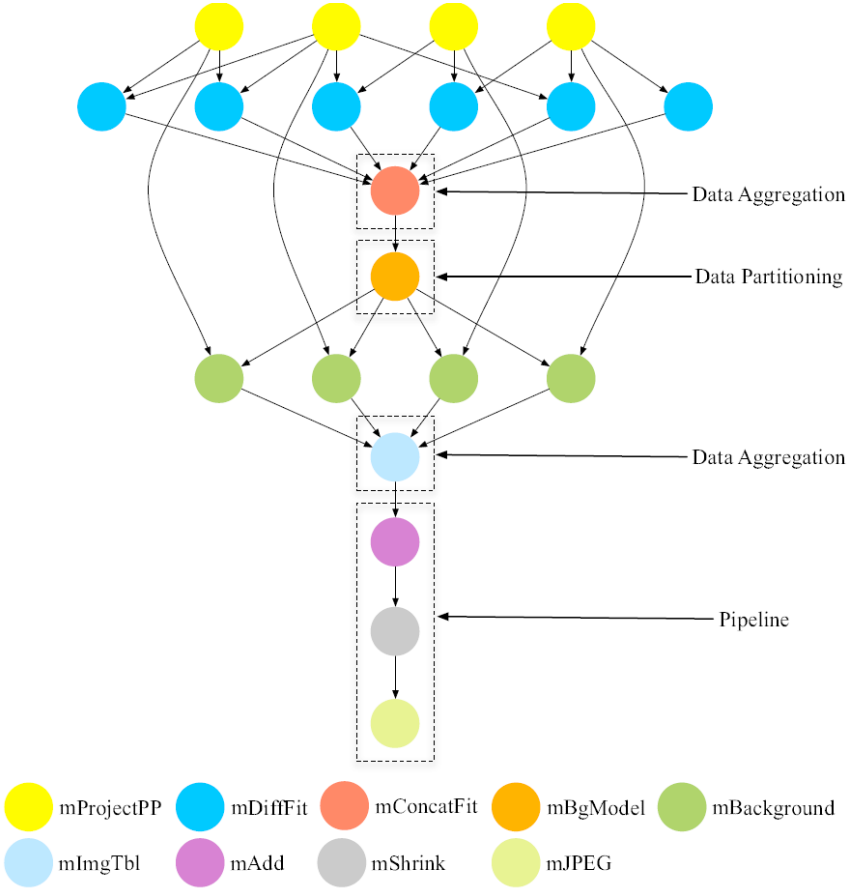

Figure 3: Montage Workflow [22].

job types of the Montage workflow are explained. The workflow consists of three basic structures discussed earlier: pipeline, data aggregation and data partitioning (distribution). The first task is mProjectPP. The amount of its jobs is dependent on the amount of FITS images given. The outputs of the first jobs are re-projected images and an 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 to mConcatFit; this is where the data aggregation is happening. After fitting the difference images it outputs the data to mBgModel, which applies a correction to each image for a better global fit – this job structure can be considered as a data distribution. The next job is mBackground, where background correction is applied to each image. After that the mImgTbl extracts the metadata from the images and creates an image metadata table which is used by several other programs. The mAdd, which is the most computation heavy job, co adds the re-projected images to form the final mosaic in FITS format and outputs also an area image. The FITS file is next reduced by the mShrink job, which outputs the shrunken file to mJPEG that converts the image to JPEG format.

2.4 Summary

The rising popularity of cloud computing has simplified the execution of scientific computing workflows for scientists. They have both free and commercial infrastructures, e.g., EC2 or OpenStack, where they can run and test their applications. With the help of scientific workflows, running and maintaining a computational application has become easier and intuitive and therefore scientists are not hold back anymore by the complexity of running an application in the cloud.

3 Partitioning the workflow

As stated before, to reduce the inter-instance communication in the deployment cluster, one 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 roughly equal sized parts, one can assign the just partitioned subsets of the workflow to run on our cluster. For example if I have a cluster containing of three instances, then it is logical to partition the workflow into three different parts. So if the goal is to reduce the inter-instance communication, then it is not efficient to run the workflow tasks randomly on top of the cluster, rather one should partition them logically before deployment. The figure 4 illustrates the basic solution to the problem. The workflow jobs are scheduled to a cluster randomly and after partitioning with METIS. As one can see, the overall sum of the weights on the cluster with three compute nodes are different when scheduled randomly (sum of 16) and with METIS (sum of 9).

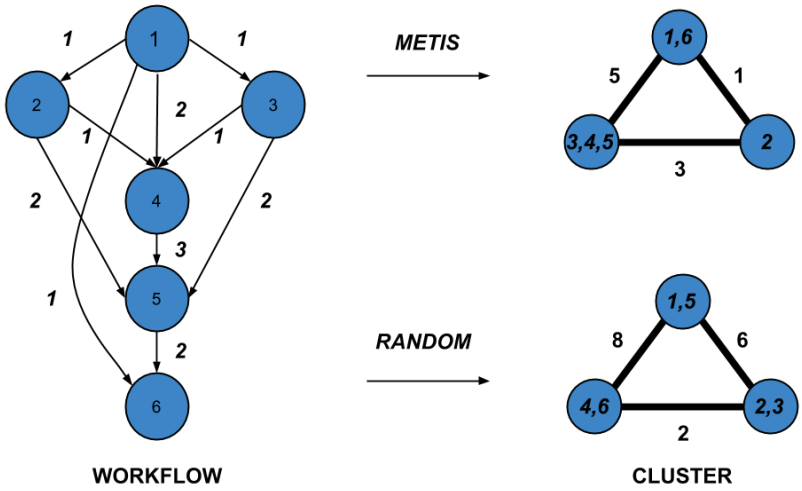

Figure 4: Differences between the communication sums after the tasks are scheduled to the cluster.

3.1 METIS

As reported by D. LaSalle and G. Karypis, graphs and graph partitioning are used in numerous areas of computing like social networks, scientific and distributed computing, biological networks, etc [23]. For that reason a toolkit named METIS has been developed, so that users can easily partition their graphs. The algorithms inside the programs are based on multilevel recursive-bisection, multilevel k-way and multi-constraint partitioning schemes that are developed in their lab Karypis.

In this thesis, the Montage workflow was partitioned with the multilevel k-way algorithm.

The METIS program takes for input a graph file, where each line, except the first one, contains the connectivity information and the weights of a particular node. The output of a graph with n vertices is a file with n lines, where on each line is an integer that shows the partition group of that node.

To partition the graph of the Montage workflow, one has to assign weights to the connections between the nodes. In this case, the weights were directly derived from the input and output sizes of each task. For example, in the 1.0 degree Montage workflow, the input of a mDiffFit job is ∼8.4 megabytes, so the weight between the nodes of mProjectPP and mDiffFit is respectively 8.4 MB. Same kind of logic was applied to the rest of the tasks to construct a weighted graph for METIS.

3.2 Partitioning

The graph partitioning problem is defined as follows: Partition the vertices of the graph in p to approximately equal partitions such that the number 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, such that $V_i \cap V_j = \emptyset$ for $i \neq j$, $|V_i| = n/k$, and $\bigcup_i V_i = V$, and the number 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, $P[v]$ is an integer between 1 and k, that shows which partition vertex v belongs to [24].

The basic idea behind of the multilevel k-way partitioning algorithm is pretty straight forward. First the graph $G = (V, E)$ is coarsened down to a small number of vertices (coarsening phase). After that the coarsest graph is partitioned (initial partitioning phase) and then the result is projected back towards the original graph (uncoarsening phase) [24]. Figure 5 illustrates the tree stages of this process. Next I describe these stages in more detail.

During coarsening phase a progression of smaller graphs $G_i = (V_i, E_i)$, is created from the original graph $G_0 = (V_0, E_0)$ such that $|V_i| < |V_{i+1}|$. The set of vertices from the graph G_i is combined to form a single vertex in the graph G_{i+1}. Let V_i^v be the vertices of G_i that are merged to a single vertex v of G_{i+1}. To maintain, that a partitioned coarser graph is also in a good ratio with the original graph, the weight of the vertex v is set equal to the sum of the weights of the vertices V_i^v. Also, in order to sustain the connectivity information of the original graph, the edges of v are the union of the edges of the vertices V_i^v. In the case where multiple vertices (more than 1) from V_i^v contain edges to the same vertex u, the weight of the vertex v edge is going to be the sum of the weight of these

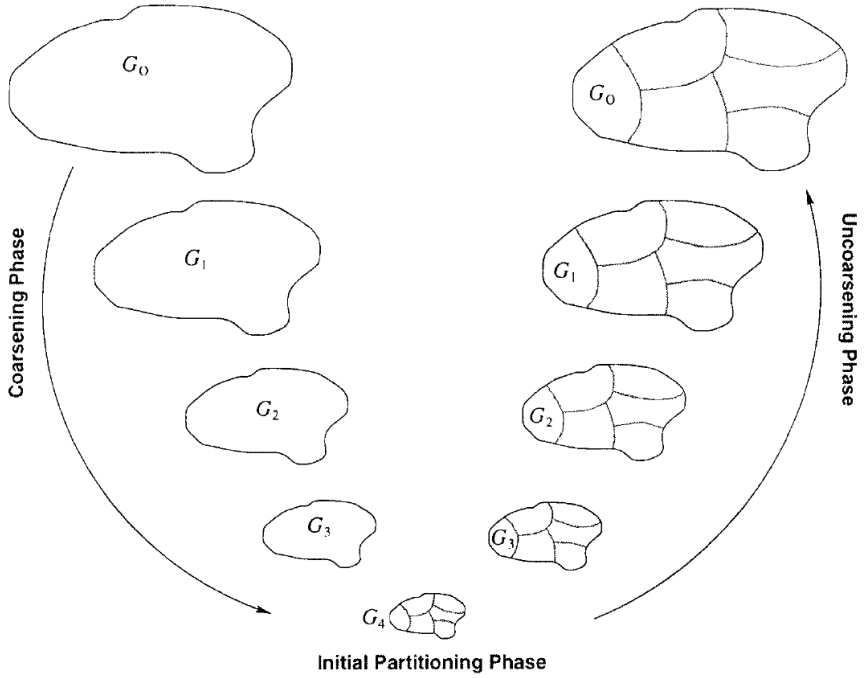

Figure 5: The various phases of the multilevel k-way partitioning algorithm [24].

edges [24].

There are several algorithms in METIS that coarsen the initial graph, e.g., random matching (RM), heavy edge matching (HEM) etc. Because the coarsening should decrease the size of the graph G_i, these algorithms are of maximal matching's. A matching of a graph can be defined as a set of edges that does not share a vertex in common. In the HEM algorithm the vertices are visited in a random order and vertex u is matched with vertex v such that the weight of the edge (u, v) is maximum over all valid incident edges. In this thesis METIS was used with sorted heavy edge matching (SHEM) algorithm, where edges are visited not randomly but in a sorted way [24].

The next stage is the partitioning phase, where a k-way partitioning P_m of the coarse graph $G_m = (V_m, E_m)$ is calculated, so that each partition contains roughly $|V_o|/k$ vertex weight of the initial graph. The k-way partitioning for the coarsest graph G_m is computed using a multilevel bisection algorithm. According to G. Karypis et.al., this algorithm produces good initial partitioning and does not take a large amount of time, as long as the size of the initial graph is sufficiently larger than k [24].

The last stage of the process is the uncoarsening phase. In this phase the partitioning P_m of the smallest graph G_m is computed back to the initial graph, by going through the graphs $G_{m-1}, G_{m-2}, ..., G_1$. Because each vertex v of G_{i+1}

consists of specific subset of vertices V_i^v of G_i, P_i is obtained from P_{i+1} by assigning the set of vertices V_i^v to the partitioining $P_{i+1}[v]$; i.e. for each u of the set of V_i^v, $P_i[u] = P_{i+1}[v]$. To improve the partitioning during uncoarsening, it is refined during the projection back to the initial graph. For refinement the METIS toolkit is using k-way refinement algorithms that are simplified versions of the k-way Kerninghan-Lin refinement algorithm [24].

3.3 Summary

To partition the Montage workflow, so that the inter-instance communication between the instances is reduced, the METIS toolkit with the k-way multilevel algorithm was used. It consists of three phases; coarsening, initial partitioning and uncoarsening phase. After partitioning the workflow subtakss are scheduled and run on the cluster for testing.

4 Approach

4.1 Pegasus

In this thesis the Pegasus application was used, which is a framework for mapping complex scientific workflows onto distributed systems [25]. In short, it is a system that schedules and submits jobs to run on top of multiple instances. Pegasus has numerous features like portability, reliability, scalability, etc. [26]. Portability means that workflows can be easily run in different environments like grids, clouds, campus clusters and so on. Reliability can be explained as an asset, where jobs that fail are rescheduled to run again, so that the probability of failure is reduced. Scalability means that Pegasus can run different size workflows on top of different type of resources.
Pegasus is composed of these next components:

- 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 and so on.

Pegasus also needs four files to run a workflow:

- Directed Acyclic Graph in XML (DAX) file – This 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.

As stated by Gideon Juve et.al, Pegasus is used to transform a resource independent, abstract workflow description into a concrete plan, which is then executed using DAGMan [29]. In short Pegasus maps the workflow tasks to available sites and then the Condor DAGMan takes over and executes the various jobs.

4.2 Condor

According to D.Wright et.al, Condor is a specialized workload management system for compute-intensive jobs [27]. It is a system that provides reliable long term computing. Users just have to submit the job to Condor and the rest like job queuing, prioritizing, monitoring, etc. are done by Condor.

Condor consists of these next main daemons: condor_master, condor_startd, condor_starter, condor_collector, condor_negotiator, condor_schedd and of condor_shadow. The condor_master is the daemon that is responsible for the starting and managing of other daemons and it runs on every machine in the Condor pool. A condor pool can be described as a 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, i.e., other daemons send updates (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. Figure 6 displays a basic Condor pool, where every machine can execute and submit jobs.

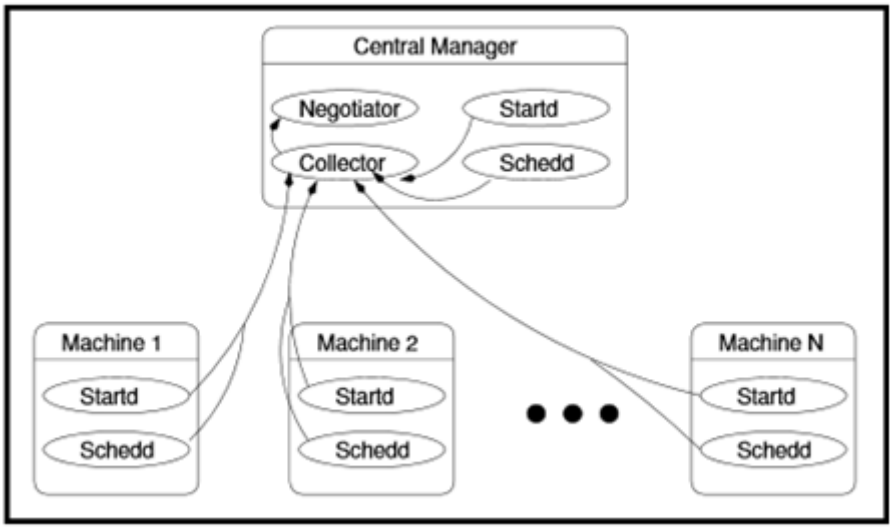

Figure 6: Daemon layout of an idle Condor pool [27].

4.3 Pegasus execution in a peer-to-peer (P2P) manner

One of the bottlenecks of running Pegasus with Condor in the cloud is the file sharing – all the communication goes through the central manager. This means that if machine 1 wants to have a file from machine 2 (figure 6), it has to send it through the central manager. This type of setup is often called a centralized file system or a NFS [28]. For these reasons, 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. Figure 7 illustrates the data transfer comparisons of Montage workflow between random scheduling and partitioning with METIS. The values are shown for three random runs where the gain with partitioning is ambiguous.

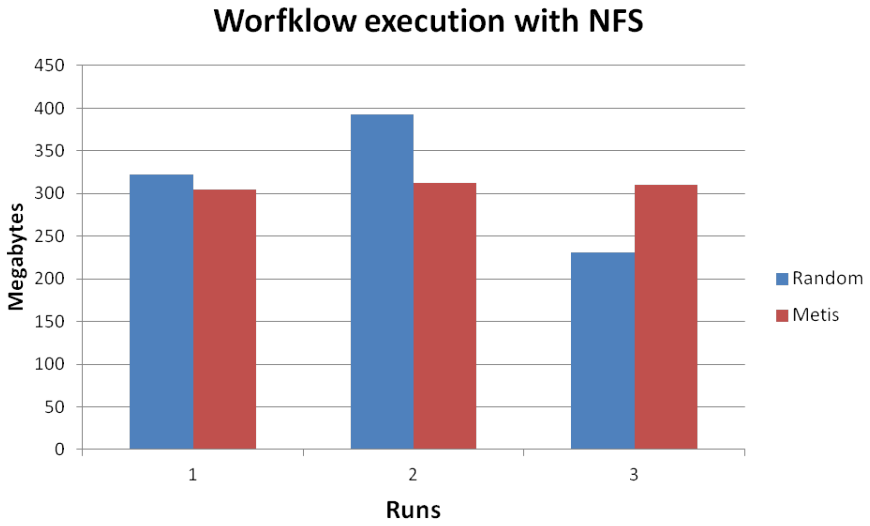

Figure 7: The data transfers in the NFS case with and without partitioning by METIS.

To solve this problem, the peer-to-peer file manager Mule [29] for Pegasus was implemented. 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 8 represents the current setup – Pegasus with Condor integrated with P2P sharing.

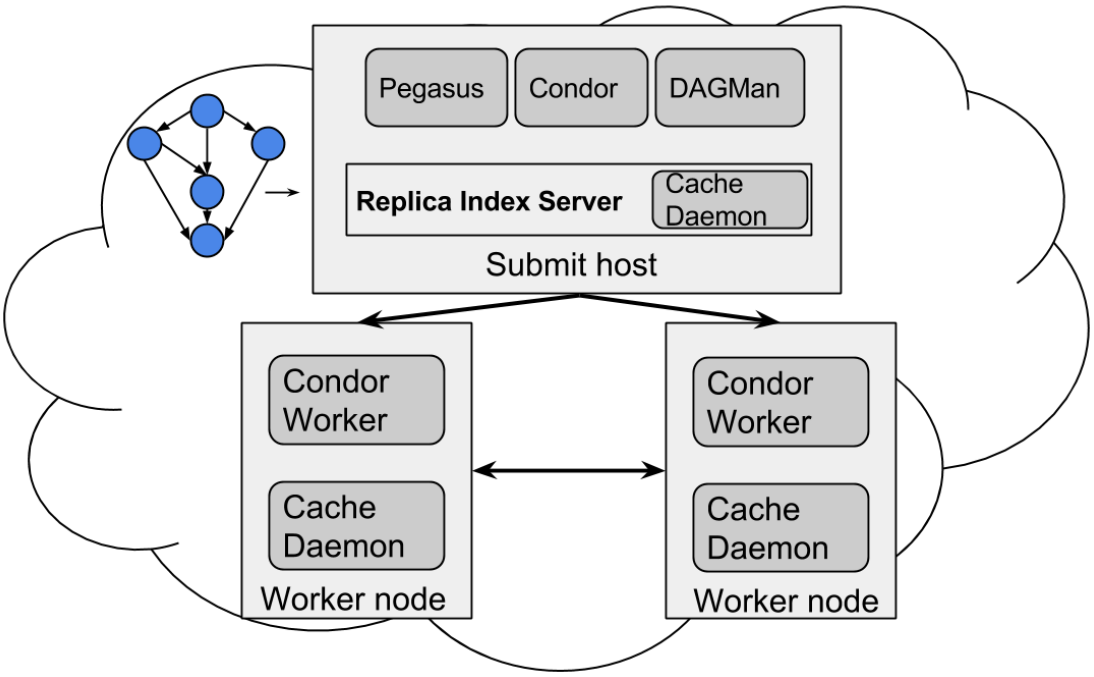

Figure 8: The current setup in the SciCloud [30] with peer-to-peer file sharing.

4.4 Setup

To run the Montage workflow, the Pegasus, Condor and Montage applications had to be installed. Condor was configured on the main node with these settings:

MAIN_IP = $(FULL_HOSTNAME), PUBLIC_IP = 192.168.100.11

DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD, STARTD

HIGHPORT = 10020, LOWPORT = 9600

NETWORK_INTERFACE = $(PUBLIC_IP)

SEC_DEFAULT_AUTHENTICATION = NEVER

On the worker nodes the MAIN_IP was changed to the submit host IP, PUBLIC_IP to the workers IP and DAEMON_LIST = MASTER, SCHEDD, STARTD, all the other settings remained the same. This configuration creates a Condor pool consisting of a central manager and worker nodes, which can execute and submit jobs.

The P2P file sharing was integrated to the Pegasus using Mule. A replica catalog server was set up on the central manager (CM) and a daemon cache on

each worker node, including the CM. Also, because Pegasus has been updated numerous times since the release of Mule, the pegasus-transfer file needed a new class:

```
class MuleHandler(TransferHandlerBase):
    _name = "MuleHandler"
    _protocol_map = ["file->mule","mule->file"]
    def do_transfer(self, transfer, attempt):
        split_url=transfer.src_url().split("/")
        filename=split_url[len(split_url)-1]
        cmd = "python /home/ubuntu/Bsc/mule/bin/mule"
        if transfer.dst_proto == "file":
                cmd += " get "+filename+" "+transfer.dst_path
        else:
                cmd += " put "+transfer.src_path+" "+filename
        try:
            myexec(cmd, default_subshell_timeout, True)
            stats_add(transfer.dst_path)
        except RuntimeError, err:
            logger.error(err)
            return False
        return True
```

In this thesis, as stated before, the Montage workflow was used. So for this reason the Montage toolkit had to be installed. Also, the transformation catalog had to contain the paths to the Montage executables for all the execution sites. For the creation of a one degree M17 nebula workflow, I used the command:

```
mDAG 2mass j M17 1.0 1.0 0.000278 . "file://inputs" "NOTREQUIRED"
```

This creates a dag.xml workflow file that contains the jobs and their dependencies. It also creates all of the other necessary files that are needed for the workflow execution, except the information catalogs (transformation, replica, sites). These were created by various scripts. To include Mule for file transferring, the sites catalog had to contain lines (in this case mainnode is the submit host):

```
    site  handle="mainnode" arch="x86_64" os="LINUX">
            <directory type="shared-scratch">
                        <file-server operation="all" url="mule://"/>
            </directory>
    </site>
```

After setting up Pegasus, Condor and Mule, the next step was to partition the Montage workflow. Figure 9 illustrates the basic process of partitioning a workflow

and mapping the result back to the original workflow.

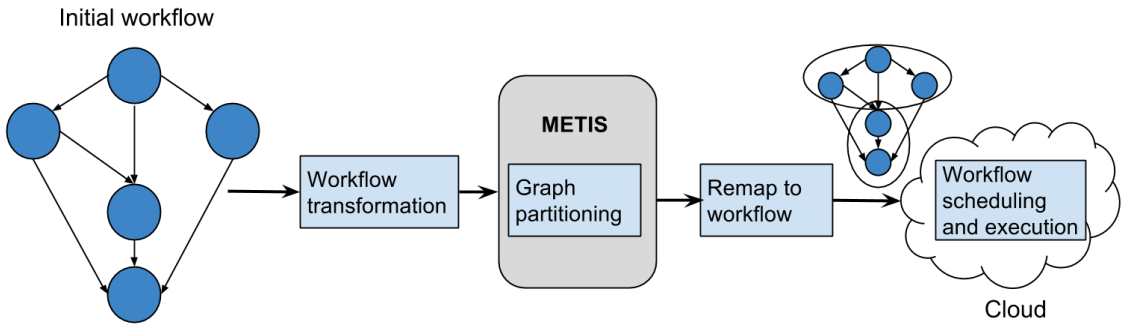

Figure 9: Partitioning process.

First the workflow file is transformed to a file format that is accepted by METIS, 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. The output of the partitioning is again a text file, where on each line is an integer showing in which group a node belongs to. The output file is transformed again to an abstract workflow file, where under each job the machine where it is going to run is specified (executionPool).

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

After the creation of the abstract workflow, where each job is assigned to a compute node, one has to plan and run the workflow. For example planning a workflow to run on top of three instances, I used the command:

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

4.5 Results

The experiments in this thesis were done with 0.2, 1.0 and 2.0 degree Montage workflows (Table 1). All of the compute nodes had one virtual CPU and 2GB of

RAM. The data transfer on the instances were measured by Wireshark [31]. The tests were run in the local cloud SciCloud [30], which is established currently with the OpenStack technology. The overall communication between the instances was reduced up to 60% with the help of partitioning the workflow with METIS.

Degrees	Data transfer sum (Randomly)	Data transfer sum (METIS)
0.2	181 megabytes	137 megabytes
1.0	1892 megabytes	770 megabytes
2.0	7290 megabytes	2741 megabytes

Table 1: Data transfer sums in the cluster when the workflow is partitioned randomly and with METIS.

Because of the implementation of the P2P file sharing on top of Pegasus alone, the overall data communication in the cluster was reduced over 50%. With the P2P approach and the partitioning the overall data communication was reduced up to 80%. For example, the data transfer of the 2.0 degree workflow case was reduced from ~15.5 GB to ~7.3 GB with the help of Mule, which further got reduced to ~2.8 GB with both the partitioning and Mule.

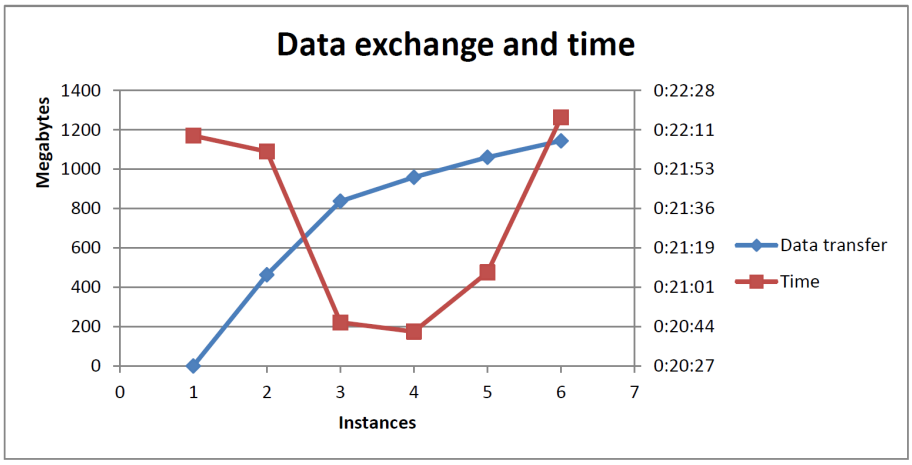

Figure 10: Data exchange and time for the completion of a 1.0 degree Montage workflow with different amount of instances.

Another interest in this thesis was do find the number of compute nodes one should provision for the completion of a Montage workflow. So, if one can find the perfect amount of partitions for one specific problem size, the same solution could be applied to other inputs of the same size. Figure 10 illustrates the data exchange and the time for the execution of a 1.0 degree Montage workflow.

In this case the amount of instances one should provision for the computation of a 1.0 degree Montage workflow, would be four, because it takes the fastest time to complete. The same applies for 0.2 and 2.0 degree Montage workflows. Though if time is not the foremost thing, then in these cases the amount of instances one should provision for the execution of a Montage workflow would be three. The reason for this is that renting an extra instance will cost more and there was not a drastic difference in the completion times when providing three or four instances.

Once the ideal cluster size had been identified, the workflow was migrated to the Amazon cloud (on 4 m1.small instances) and executed for the 2.0 degree Montage. Figure 11 shows the data communication latencies, for both the cases where workflow is scheduled randomly and where it 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%.

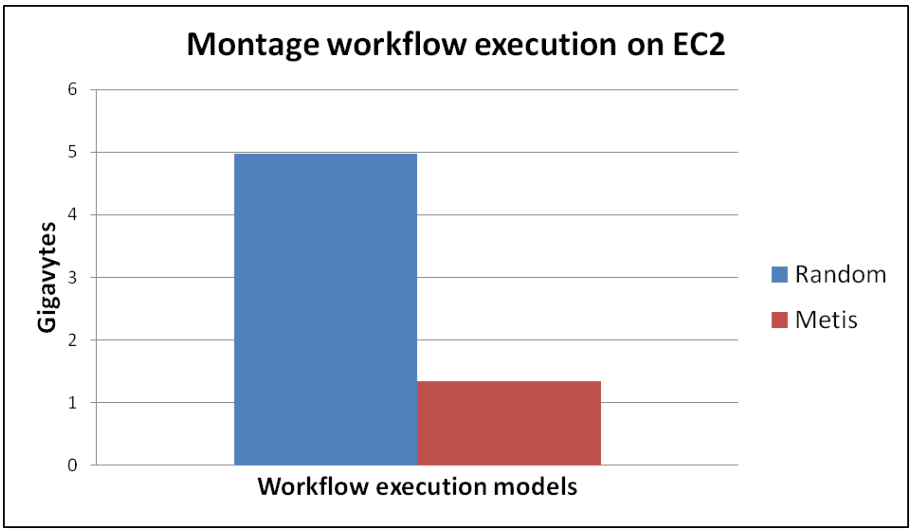

Figure 11: Data communication across 4 nodes in Amazon EC2 while performing the execution of 2.0 degree Montage workflow.

4.6 Summary

For the scheduling and management of scientific workflows in the cloud the Pegasus application was used. Because the partitioning did not work in the centralized file system case, the peer-to-peer model was included. With the P2P implementation and the partitioning the communication was reduced in the computation cluster up to 80%. The ideal cluster size was also found for the 0.2, 1.0 and 2.0 degree Montage workflows, which was either three or four, depending upon preferences.

5 Conclusions

In this thesis a scientific workflow was partitioned to reduce the communication between the instances in the cloud. The partitioning was done with the METIS library. The workflow management and execution was done with Pegasus.

Initially the partitioning did not work, because all the communication went through a central node, which stored all the necessary data. This type of setup is often called a centralized file system and is more common in grid environments and in high-performance computing.

For these reasons the peer-to-peer model Mule was used on top of Pegasus, which allowed all of the computational nodes to share the necessary files with each other. The peer-to-peer implementation itself reduced the communication between the computation nodes alone nearly by 50%.

To reduce the data transfer even further, the workflow had to be partitioned, in order to minimize the sum of the weights of the edges connecting to vertices in different groups. This task was done with the help of METIS.

First the abstract workflow format was transformed to a file that was readable by METIS. After the transformation, the workflow was partitioned with METIS and the output remapped back to the original abstract worfklow. After partitioning the scientific workflow and rescheduling the jobs on to the compute sites, the data transfer was reduced up to 60%. In short, the peer-to-peer approach with the partitioning reduced the communication in the computation cluster up to $\sim$80%.

Another interest in this thesis was to find the optimal amount of computational nodes one should provision for the completion of 0.2, 1.0 and 2.0 degree Montage workflows. For all these cases, the reasonable amount of instances to provide was either three or four, depending whether the emphasis was more on the cost or the time. After finding the ideal cluster size, a 2.0 Montage workflow was migrated to the Amazon cloud, where the data communication was reduced after partitioning nearly by 70 %.

In conclusion, all goals that were set for this thesis were achieved. The concept, observations and results were also formulated to a scientific paper that was submitted to the HPCC2014 conference.

6 Future research directions

Regarding the future work, the discussed partitioning methods results 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.

Teaduslike töövoogude modelleerimine pilve jaoks

Bakalaureuse töö (6 EAP)
Jaagup Viil

Resümee

Viimastel aastatel on üha enam inimesi hakanud kasutama pilve poolt pakutavaid teenuseid. Ettevõtted ei pea enam raha kulutama süsteemide administratsioonile, riistvara parandamisele jms. Vajalikud ressursid saab lihtsalt rentida pilve teenuste pakkujatelt ning maksta tuleb vastavalt palju antud teenust kasutatakse nagu elektri või vee kasutamise korral. Üks suurimaid pilve eeliseid on ressursside elastsus. See tähendab, et ressurside arv mida renditakse ei ole limiteeritud ning seda saab vastavalt vajadusele juurde hankida.

Kui pilve poolt pakutud teenuseid kasutavad enamus ajalt ettevõtted, siis on see kogumas populaarsust ka teadusarvutamise vallas. Tavapäraselt kasutatakse selles valdkonnas teaduslikke töövoogusid, mis üldiselt koosnevad suurest hulgast töödest ning seetõttu vajavad ka palju arvutusressursse, mida just pilv võimaldabki pakkuda.

Varasemad uuringud näitavad, et üks suurimaid probleeme teadusprogrammide jooksutamisel pilves on omavahelise andmevahetuse suurus. Üks lahendus sellele probleemile oleks tuvastada komponendid, mis omavahel palju suhtlevad ning panna nad pilves ühte kohta jooksma, et vähendada omavahelist andmevahetust.

Antud bakalaureuse töös jagatakse ühe kindla teadusprogrammi osad pilves asuvate virtuaalmasinate vahel, et vähendada omavahelist andmevahetust. Et teada, kuidas jaotada programmis olevad tööd masinate vahel nii, et suhtlus nende vahel väheneks, kasutati graafi teooriat. Algus programmi partitsioneerimine (osade jaotamine) ei vähendanud üldist andemevahetust, kuna kogu suhtlus käis läbi ühe kindla virtuaalmasina. Selle probleemi lahendamiseks rakendati valmis kirjutatud P2P süsteemi, kus iga masin saab iga teise masinaga suhelda. Juba P2P rakendamine vähendas ligi 50% andmesuhtlust pilves. Pärast programmi partitsioneerimist ning tööde jooksutamist vähenes kommunikatsioon pilves veelgi 60%. Kokkuvõttes võib öelda, et tänu P2P süsteemile ja teadusprogrammi partitsioneerimisele vähenes kogu suhtlus pilves kuni 80%.

References

[1] 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.

[2] S. N. 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.

[3] 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, pp. 359–392, 1999.

[4] Montage, "An astronomical image engine," visited 20.03 2014. [Online]. Available: http://montage.ipac.caltech.edu

[5] 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.

[6] SIPHT, visited 23.03 2014. [Online]. Available: http://newbio.cs.wisc.edu/sRNA/

[7] Ü. V. Çatalyürek, K. Kaya, and B. Uçar, "Integrated data placement and task assignment for scientific workflows in clouds," in *Proceedings of the fourth international workshop on Data-intensive distributed computing*. ACM, 2011, pp. 45–54.

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

[9] 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.

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

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

[12] 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.

[13] Amazon Inc., "Amazon elastic compute cloud (amazon ec2)," visited 6.04 2014. [Online]. Available: http://aws.amazon.com/ec2/

[14] OpenStack, visited 14.04.2014. [Online]. Available: http://www.openstack.org/

[15] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, Nov 2008, pp. 1–10.

[16] D. Kahanwal, D. T. Singh *et al.*, "The distributed computing paradigms: P2p, grid, cluster, cloud, and jungle," *arXiv preprint arXiv:1311.3070*, 2013.

[17] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on.* Ieee, 2008, pp. 5–13.

[19] A. Zahariev. (2009) Google app engine.

[20] 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.

[21] D. D. R. C. G. Katy Wolstencroft, Paul Fisher, "Scientific workflows," 2009. [Online]. Available: http://cnx.org/content/m32861/1.3/

[22] 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, 2008. WORKS 2008. Third Workshop on.* IEEE, 2008, pp. 1–10.

[23] 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.

[24] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48(1), pp. 96–129, 1998.

[25] 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.

[26] Pegasus, 2014, visited 8.04.2014. [Online]. Available: http://pegasus.isi.edu

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] G. Combs *et al.*, "Wireshark," *Web page: http://www. wireshark. org/last modified*, pp. 12–02, 2007, visited 25.04.2014.

License

I, Jaagup Viil(date of birth: 05. April 1992),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Remodelling Scientific Workflows for Cloud,
supervised by Satish Narayana Srirama,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual proper-ty rights or rights arising from the Personal Data Protection Act.

Tartu, 14.05.2014