Network Topology Analysis in the Cloud

Thomas Mundt and Jonas Vetterick
University of Rostock, Germany - Department of Computer Science
thomas.mundt@uni-rostock.de

***Abstract*—This paper demonstrates how Cloud resources can be used for scientific calculations in a cost effective manner. The target audience consists of researchers who have to calculate a large amount of data. The paper does not deliver new scientific results in the traditional way in the area of Cloud computing, but explains the economic and technical advantages and disadvantages of Cloud computing for use in science. It is meant to foster the use of cloud resources as a cost effective way to run scientific analyses. For this, a cost estimation is given for an example implementation. A running example originating in the research area of computer network optimization is explained from the technical background to the final implementation.**

***Index Terms*—Cloud computing, network analysis, scientific computing.**

I. Introduction

During empirical studies researchers typically have to analyse large amounts of data. In this paper we use an actual calculation needed in an ongoing scientific project to investigate if commercial cloud providers could be an alternative to local high performance computers or computing clusters. The calculations used as case study take time series of network parameters originating from a large wireless mesh network as input and deliver several quality indicators.

The remainder of this paper is organized as follows. Section II explains the technical and functional background of the example application used in this paper, section III compares two possible cloud service levels and gives a short overview about available commercial and academic cloud providers. In section IV the example implementation is discussed as well as costs and performance are assessed. Section V concludes this paper and suggests further activities.

II. Functional background of the example application

This section explains the functional background and gives and overview about needed calculations and data sets collected for examination.

A mesh network basically consists of nodes that share a wireless communication channel. Every node in the network forwards data packets for other nodes according to a routing scheme to its neighbours. This decentralized structure allows cost-effective networks but requires a sophisticated routing mechanism.

The mesh network [7] under investigation uses Optimized Link State Routing (OLSR) [12]. OLSR is a pro-active link-state routing protocol. The mode of operation of pro-active link-state routing protocols requires that current information describing the complete topology is available at every node before data packets are being sent. Hence, a large amount of topology information have to be permanently distributed through the entire network. Using this topology information all nodes are able to calculate paths for outgoing data packets themselves.

In OLSR every router broadcasts topology control (TC) messages with a pre-configured update rate. These TC messages consume a non-neglectable share of the available throughput and generate interference in a larger mesh network. The final goal of the parent research project is to reduce the update rate and pre-compute topology changes, which might be caused by recurring events, such as human behaviour (daily routine) or natural phenomenons.

A. Data set

We use historic data describing the topology at certain times to analyse those phenomenons. For this purpose the routing situation at every time stamp in a long term time series needs to be reconstructed. TC messages are available at every node participating in the network. Delays caused by the propagation of TC messages are inevitable. Hence, all data represent the current situation at the point within the network where all data is being collected.

According to OLSR principles every link between two nodes is attributed by two parameters, link quality (LQ, from current node to neighbour node) and neighbour link quality (NLQ, reverse direction). Note that links are not considered to be symmetric. Both values describe the probability that a single data packet reaches the node at the opposite side of the link. Probe packets ("Hello" messages) are used to determine the link quality.

Data containing LQ and NLQ values for every available link have been snapshotted every minute for several months. The routing algorithm uses this topology information and chooses the best paths from source to destination according to LQ and NLQ values. More detailed, the expected number of transmission attempts needed before a packet reaches its destination (expected transmission count - ETX) is used as metric.

Currently the data collection consists of about 195 million tuples each representing a single link at a certain point in time. [1] The data base containing these 195 million tuples has a size of about 22 gigabyte. About 240.000 snapshots (distinct time stamps) are available by now. On average there are 180 nodes

[1]All data and software used in this paper is available for download from open science repository co-maintained by the authors at http://opsci.informatik.uni-rostock.de/.

and 810 available links forming the mesh network at every point in its history. Each tuple represents one link at a time and has the following structure:

Attribute	Meaning
timestamp	Timestamp of snapshot, same for all links being active at the time of taking the snapshot.
thisNode	Name (IP address) of local node
otherNode	Name (IP address) of remote node
lq	Link quality from thisNode to otherNode (ratio of received OLSR "Hello" messages to sent probe messages)
nlq	Link quality from otherNode to thisNode

TABLE I
MEANING OF DATA ATTRIBTUES.

A short example for several links looks like this (II:

timestamp	thisNode	otherNode	lq	nlq
2010-09-10 00:00:02	B 192.168.1.129	A 192.168.0.254	0.780	0.800
2010-09-10 00:00:02	B 192.168.1.129	E 192.168.1.14	1	1
2010-09-10 00:00:02	B 192.168.1.129	D 192.168.1.15	0.450	0.450
2010-09-10 00:00:02	E 192.168.1.14	D 192.168.1.15	1	1
2010-09-10 00:00:02	E 192.168.1.14	C 192.168.1.2	0.800	0.800
...	...	...	...	...
2010-09-10 00:01:02	B 192.168.1.129	A 192.168.0.254	0.820	0.800
2010-09-10 00:01:02	B 192.168.1.129	E 192.168.1.14	1	1
2010-09-10 00:01:02	B 192.168.1.129	D 192.168.1.15	0.550	0.500
2010-09-10 00:01:02	E 192.168.1.14	D 192.168.1.15	1	0.980
2010-09-10 00:01:02	E 192.168.1.14	C 192.168.1.2	0.990	1
...	...	...	...	...

TABLE II
EXAMPLE DATA SETS DESCRIBING SEVERAL LINKS AT TWO DIFFERENT TIMESTAMPS.

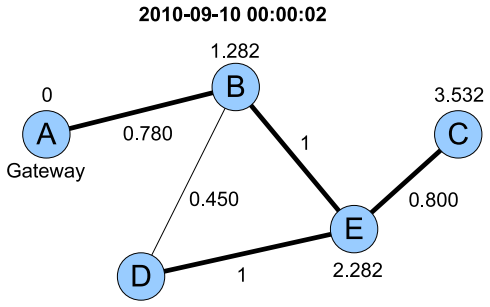

Fig. 1. Network condition at a point in time. For simplicity only LQ values are attached to links. Bold lines show used links.

B. Calculations

All tuples in a snapshot (those tuples carrying the same time stamp) together describe the network topology as seen by the node where all data are being collected. The network has several gateways to the Internet. Internet usage is by far the main usage of the mesh network. TC and "Hello" messages are broadcast by every node without routing. So, basically all other traffic is routed towards Internet gateways and in reverse direction towards the nodes directly serving users. The quality of the path between user node and gateway determines

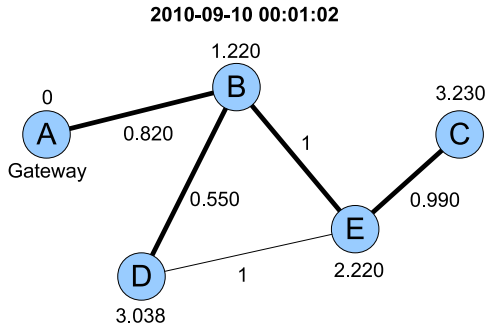

Fig. 2. Network condition at another point in time. For simplicity only LQ values are attached to links. Bold lines show used links. Routing has changed due to a change of link quality.

the quality of the Internet connection. There have been four gateways during the data collection campaign. Every node is assumed to connect to the nearest gateway in terms of ETX metric.

With computing the path quality to and from all available gateways for all active nodes and for every snapshot we hope to find recurring patterns usable for later predicting the network status. For this purpose the path quality to all gateways is being computed. At first, the shortest paths according to the ETX metric is calculated towards all gateways with the traditional Dijkstra algorithm. This delivers the same paths as for real data packets during at the time of data collection. As result every node is labeled with four path ETX values. Additional quality describing values such as hop count to the gateways, and path link quality are also computed.

The following estimation illustrates the complexity of calculations. The database contains 240.000 snapshots (growing). Every snapshot contains 810 available links. Most of them are never becoming active links, but all of them have to be considered when the shortest paths are calculated. Those 810 available links connect about 180 active nodes. There are four gateways to be independently considered. For every node the shortest paths (in terms of the ETX metric) to all four gateways have to be computed. The path ETX has to be stored with every node for each of the four gateways. The minimum ETX among those four ETX values represents the node's connection quality towards the Internet.

Calculating path ETX values for one snapshot and one gateway takes about 900ms on a local computer (Intel i7 CPU, sufficient RAM to hold all data, algorithm implemented in Java and python). For all 240.000 snapshots and four gateways it would take an estimated 240 hours computation time ($0.9s \cdot 240,000 \cdot 4$). The process as a whole has to be repeated when routing parameters or other aspects of the analysis are changed, as if they are used in a simulation. This is was not acceptable for a detailed analysis. The calculation process is parallelizable (distributable) in several dimensions. Therefor the calculation time can be decreased by using more hardware. As a feasibility study a cloud based implementation was preferred over a local cluster.

III. Cloud computing used for research

This section introduces relevant concepts of cloud computing. It is not meant to give a full overview about cloud providers, but shows significant advantages and possible pitfalls.

The term "Cloud computing" as used in this paper refers to computing resources available on the Internet. In most cases resources are leased to customers. Several levels of services are available for leasing. Applications are deployed to unknown locations. "Cloud computing" is a new form of distributing software, using resources, and delivering services on the Internet. A major advantage of using clouds is that applications are highly scalable since most cloud computing infrastructures deliver services through a network of computing centers and built on a large number of servers. These servers can be used in a distributed way for scientific calculations when algorithms are parallelizable.

From an economic point of view cloud computing could decrease investment costs for computational complex analyses and simulations, especially when computing power is needed for short periods only. In these cases building own infrastructure that idles most of the time is uneconomic.

Cloud providers can deliver services on different levels, which are usually referred to as Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Software as a Service (SaaS). PaaS and IaaS are most suitable for scientific problems as they facilitate deployment of applications without the cost and complexity of buying and managing the underlying hardware and in case of PaaS also the underlying software. A more general overview about cloud technologies and different service levels is given in [21]. Concrete examples can be found in the following sections.

A. Platform as a service and Infrastructure as a service

As mentioned before, mainly two service levels are available, Infrastructure as a service and Platform as a service. Basically both are suitable for the example network calculation described in this paper.

The main difference is that using Infrastructure as a service requires to set-up a computer as it would happen to a local machine, including operating system, libraries, and software, while using a Platform as service requires deploying of software to the platform only. But, the latter approach limits resources to those provided by the platform.

Amazon EC2, as an example for Infrastructure as a service, provides several pre-configured images as a starting point for configuration. After installing the software the prepared image can be started several times. This is controlled by the user through web services. Each instance works with its own separated resources. Amazon EC2 provides different instance types (machines) with different hardware configurations, for example high memory, high CPU, or cluster compute instances. The granularity of adding and removing resources is limited, since an entire instance has to created or destroyed at once. Additionally, different quality of service levels can be purchased, for instance the so-called spot instance which only runs when the spot price for computing capacities is below a threshold. This is useful since scientific calculations as described here could be computed during off-peak times in order to save money.

Google App Engine, as an example for Platform as a service, hides the infrastructure from users. Instances of user applications are created on-demand. Separate virtual machines are indistinguishable for users and applications, and, hence, neither have to be set-up, created, or destroyed. Resources are provided to each application instance. The application developer has to regard the APIs provided by the platform. Additional resources beyond those API are not available. In the concrete case of Google App Engine the service is provided through HTTP, which imposes limits on calculation time.

A general recommendation for the application of either PaaS or IaaS for scientific computing cannot be given. IaaS offers more variability while PaaS reduces the effort to maintain the system. General criteria are price, performance, and ease of development and use. The task of calculating network parameters used as example in this paper could be fulfilled with both designs.

B. Scientific and commercial cloud providers

A brief overview about commercial cloud providers can also be found in [8]. It is obvious that major Internet companies also provide cloud services. This is reasoned by the history of cloud computing. Most commercial providers of the first generation have developed cloud services to sell their excess capacity in computing centers to others.

Larger commercial cloud providers are Amazon Elastic Compute Cloud (EC2), Google App Engine (GAE), and Windows Azure (WA). All of them offer services on a per-use basis, which is common for cloud computing.

Although universities and other research bodies maintain large computer farms and although several initiatives have been started to provide cloud services for scientific calculations currently no cloud-like computing environments or only prototypes are available for external researchers.

Scientific cloud computing is discussed since around 2008 [25]. For this purpose both non-commercial and commercial cloud providers are considered [14] [17] [16]. Examples for non-commercial initiatives in North America are ScienceClouds [24] and a NSF project [15], [27]. ScienceClouds, for instance, is able to provide cloud services at the same interface as Amazon Elastic Compute Cloud.

C. Available cloud computing resources

For the purpose of analysing large amounts of data the Cloud provider must at least offer facilities to compute and store data. Input-output-functionality is needed for transferring data into the cloud and downloading results.

Amazon EC2 and Windows Azure mainly provide a basic infrastructure. Typically this includes a set of virtual hosts. On those hosts users can install software as needed. Typically file system images are used for this. Scaling the number of virtual hosts up and down is a basic function of the cloud to adapt

resources to changing needs. Developers can use memory, CPU, and network connections on a virtual system. They are free to install frameworks and use programming languages at their discretion. In case of Amazon EC2 developers can choose among several operating systems and different instance types. Instance types are available from so called micro instances up to cluster instances [1].

Persistent storage has to be provided by the developer through conventional relational database management systems. Alternatively, Amazon Simple Storage Service (S3), Amazon SimpleDB, or Amazon Elastic Block Store (EBS) can be used, all three are very well integrated with Amazon EC2.

With Eucalyptus [20] an open source platform is available for the implementation of private cloud computing on computer clusters. This means, Eucalyptus allows owners of hardware to provide Infrastructure as a service.

Contrary to that, Google App Engine provides a closed platform which hosts the final application. The platform consists of a set of APIs which enable the application to use a variety of services such as authentication, fetching web pages via HTTP, sending e-mails, and persistence of data. Applications have to be written in Python or Java and will be executed in a sandbox. Data storage is provided by Google through its Datastore service [9]. Applications are deployed and run as web service or servlet, both of them providing an HTTP interface. This limits compute time to one request-response-cycle.

The example used in this paper (calculating routes and network quality values) can be easily split into small sub-tasks. For this reason the time limit for processing HTTP requests is not an issue. An alternative way to deal with that limitation are task queues [22]. A task queue collects background work that is organized into smaller units. Tasks are executed when cloud provider system resources permit.

An open source platform (PaaS) that is compatible with Google App Engine and which can be deployed into an IaaS cloud such as Amazon EC2 or Eucalyptus (see above) is available with AppScale [10].

D. Data storage and data retrieval

Traditional database management systems can be used within Amazon EC2 and Windows Azure beside file based concepts whereas Google App Engine only provides a non-relational (NoSQL) [19] [18] storage concept called Google datastore [11]. The advantage of those storage is that it can easily be distributed among instances. A disadvantage is that queries are limited to selections (choosing data sets). Projections (choosing a set of attributes) and aggregations (for instance calculating sums or grouping values) are not permitted. Application developers should be aware of those disadvantages when designing software. In some cases operations such as grouping values have to be performed within the application itself instead of leaving this to the database.

In the concrete example the datastore itself could not deliver a list of timestamps since this would require to group several data sets by distinct attributes. Building this list require either a separate table (actually it is not a table in the sense of SQL because not all attributes have to be present for all data sets) could be used or the entire datastore would have to be read.

Scientiest should also consider to upload their data into public data sets, such as discussed in [4]. This collection of data is sponsored by Amazon and made available to cloud applications via the Amazon S3 API.

E. Application control, security, and costs

Cloud users have to control the resource usage of their applications. For scientific computing this includes to set the number of instances, access rights, scheduled tasks, and financial limits. The framework reports for instance the current load, throughput, amount of used and remaining storage space, used CPU time, and all other billable resources. Google App Engines for instance provides a web based control center, see Figure 3.

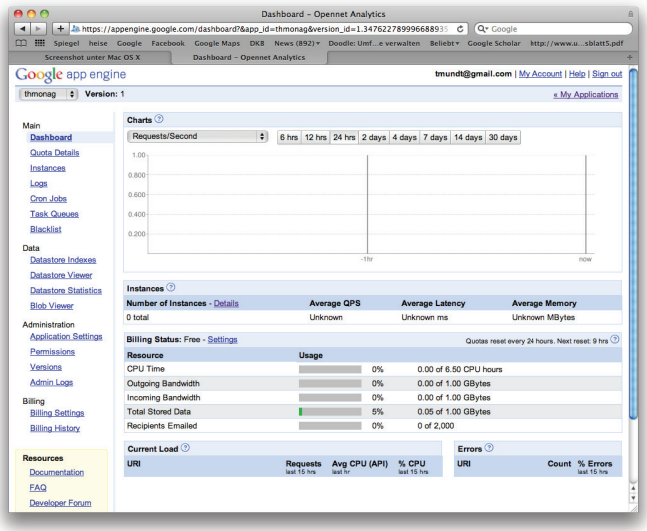

Fig. 3. Google App Engine "Dashboard" reporting current load and billing information.

All cloud providers bill for CPU usage and data storage. Further billable resources include transactions, data transfer, API calls such as for sending and receiving e-mails, building datastore indexes and for image manipulation.

Tables III and IV show examples for pricing models. Please note that CPU hours not necessarily comparable between different systems, but these figures give a hint about real costs.

Cloud provider	Costs per CPU and hour
Amazon EC2	8.5 US-Cent
Google App Engine	10 US-Cent
Windows Azure	12 US-Cent

TABLE III
COSTS FOR CPU USAGE. [2] [5] [6]

Several payment methods are available. Using resources subject to charge requires a credit card or bank account. Cost

Cloud provider	Costs per GB storage space and month
Amazon EC2	14.03 US-Cent
Google App Engine	15 US-Cent
Windows Azure	15 US-Cent

TABLE IV
COSTS FOR STORAGE USAGE. [2] [5] [6]

limits can be set for each application. Applications can easily be deployed via HTTP. Setting up a new application can typically be realized within a few minutes.

Cloud computing environments bear several risks in terms of availability and data security. Generally data security is an issue when applications and data are distributed to the cloud. Although cloud providers promise confidentiality a certain risk of technical problems remains. Availability of resources raises further concerns. Although the short history of cloud computing showed that outages are very rare critical applications should not be deployed to a cloud provider. The sheer amount of redundant resources decreases those risks dramatically.

F. MapReduce

MapReduce is a technology invented by Google to handle large amounts of data [13]. It works by performing a two stage computation. In the first stage (Map), the input is divided into smaller subsets and distributed to so-called worker nodes. A worker can either process the smaller subset or divide it into further subsets. The latter case would lead to a tree structure. In the second stage (Reduce) the answers from all workers are combined to get the output. This is exactly what is needed to perform the calculations in the example.

Hadoop [26] is an open source implementation supported by the Apache Foundation. Hadoop is, for instance available in pre-configured Amazon EC2 instances under the name Amazon Elastic MapReduce [3].

IV. EXAMPLE IMPLEMENTATION OF NETWORK ANALYSIS SOFTWARE

In order to understand the following cost and performance estimations an overview about relevant components of the application are given in this section. The focus is on PaaS(concretely Google App Engine). An IaaS based approach would have the same basic architecture, except that the platform on top of the infrastructure has to be provided by the application developer. Aspects such as consistent storage are not relevant for the concrete example application.

A. Software architecture

In this section all concepts are named by their respective name within Google App Engine. Other cloud provider use different names for similar concepts.

The application is very simple, hence, the architecture remains very straightforward. It needs to store data and performs many calculations where the same data is used multiple times. For this reason all data is stored in the cloud in order to increase access speed. A major reason is that data sets are replicated transparently through the scaling mechanism maintained by the cloud provider. Alternatively, data could be stored in the local data center and being transferred on demand, but this would decrease access speed significantly.

Figure 4 shows a general overview about the layers used to access the datastore and to compute the results for the example used in this paper. Of course, other APIs can be called as well, but are not used in the example.

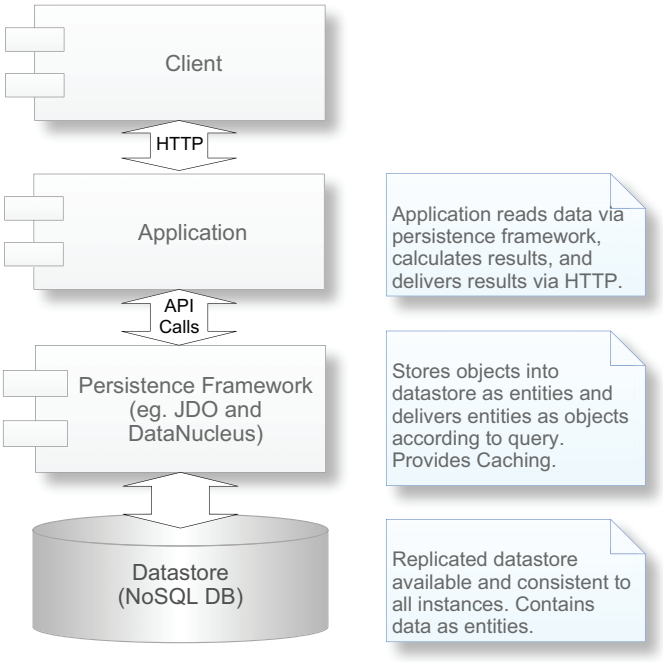

Fig. 4. Components forming the sample application on top of Google App Engine.

The datastore keeps data objects with their persistent attributes as entities. These entities are comparable to tuples in relational databases, but the main difference is that each entity can have arbitrary named values. The result is not a table but a collection of entities of a kind. Two entities of the same kind can have different properties. Hence, the Google Query Language (GQL) does not support projections and aggregations. Several data types are supported, for instance string, integer, or references to other entities. Distributed transactions, which occur when multiple users access the same data at the same time, are supported by the datastore. The datastore uses a distributed architecture to scale. Furthermore, customers can choose between a master/slave replication where one master copy of data is kept in one of Google's data centers, or high replication (at a higher cost).

It is common that application classes do not access the database or datastore directly through a low level interface. Such a low level interface is provided by Google, but more sophisticated interface is also availabe. In case Java is used as platform, persistence is provided through the DataNucleus project [23] which finally provides a JDO implementation to the application. This eases datastore access dramatically.

Basically all persistent attributes of the entity class have to be labeled as “Persistent” such as in this example:

```
@PersistenceCapable
public class Link implements Comparable<Link> {

  @PrimaryKey
  @Persistent(valueStrategy =
    IdGeneratorStrategy.IDENTITY)
  private Key key;

  @Persistent
  private Date timestamp;

  @Persistent
  private String thisIP;

  @Persistent
  private String otherIP;

  @Persistent
  private double lq;
  ...
```

Particular entities or sets of entities can be queried from the datastore through a persistence manager which accepts GQL or JDO syntax.

Calculations can be triggered by HTTP requests. Results of calculations can be stored in the datastore or can be delivered to the client via HTTP, as HTML or XML for instance. An XML web service would be another suitable bidirectional interface.

For the example application results are put into an XML document like this:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="/calc.xsl"?>
<timestamp time="Fri Sep 10 00:00:02 UTC 2010">
 <root name="192.168.0.254">
  <calctime>1707</calctime>
  <node>
   <sequencenumber>1</sequencenumber>
   <name>10.2.0.247</name>
   <quality>8.056671</quality>
   <parent>192.168.1.27</parent>
  </node>
  ...
  <node>
   <sequencenumber>2</sequencenumber>
   <name>10.2.0.251</name>
   <quality>10.0566718</quality>
   <parent>192.168.1.130</parent>
  </node>
 </root>
</timestamp>
```

B. Software development process

The software development process for PaaS clouds is very straightforward, especially when the frameworks provided within the PaaS cloud are well established in the off-cloud world. This is the case with the Java environment provided by Google App Engine. In an IaaS cloud the software development process is comparable to regular server based system.

A local testing environment is preferable, as it limits deployment efforts and shortens the development and test cycle. The completed application can than be released to the cloud environment. Deployment of applications (Google App Engine) or instance images (Amazon EC2) is supported through helper applications or plug-ins for IDEs, such as Eclipse. Although very straightforward, deployment consumes a certain amount of time.

Debugging an application on a PaaS cloud is a hassle, since step-by-step execution or console output is generally not available and logging remains the only way to inform the developer about the current state of the software. Therefore, local testing of components is inevitable.

V. Test results, costs, and conclusion

Each computation step (calculating all paths to one gateway for one timestamp) consisting of fetching the data for this timestamp, building the internal model, performing the Dijkstra algorithm, and calculating. The entire calculation explained in section II takes about 13 hours, which is approx. 18 times faster than computation on high end local resources (approx. 240 hours). To protect their infrastructure, cloud providers limit the number of possible instances. Google App Engine for example creates up to 20 instances when many requests are queued and consequently workload is high. The factor of 18 fits very good with the number of available instances considering some overhead.

For this calculation 750 CPU hours have been charged, which results in costs of 75 US-Dollar.

The entire dataset needs around 48GB when stored in the datastore (data plus meta-data). This costs additional 7.20 USD per month. The amount of uploaded data is about 22GB (note, that all data are uploaded via HTTP which imposes an overhead). Uploading of these data costs 2.20 USD one time for data transport and approx. 2 USD for CPU time (eg. creation of indexes). Uploading can and should be parallelized as well through multiple instances, otherwise uploading data takes several days time.

Some figures shall be given for a very rough comparison. A local machine would need 10 days. The total cost of ownership for a server system in the University’s computing center the authors are using is estimated at around 3000 EUR per year (750 EUR amortization of hardware over 4 years, 700 EUR electricity, 100 EUR external networking, 25 EUR rent for rack space incl. climate control etc., 1400 EUR administration). At an idealized full capacity this would occasion costs of around 80 EUR per 10 days or 110 USD (at an exchange rate of 1.40 valid in February 2011).

The problem described in this paper as a running example is very well parallelizable. For this reason it benefits much from additional computation resources. Additional resources can be provided through cloud computing or through local hardware. Local hardware needs to be working to maximum capacity to decrease costs. In comparison, cloud computing is more cost effective when resources are needed at peak times or the workload is unevenly distributed over life time.

Generally, the following conclusions can be drawn from the example project:

- Cloud computing is a suitable option for scientific computing. For calculations that require shorter peaks of computation time cloud computing is a cost-saving option.
- PaaS supports instant software development without the need of installing operating systems and runtime environment. This is subject to the availability of all needed functions within the platform.
- Availability of cloud resources (both long term and short term) should be considered when larger software development efforts are necessary.
- Cloud computing concepts are suitable for resource sharing withing the academic community. This would increase the load factor and decrease costs.

REFERENCES

[1] Amazon EC2 Instance Types. http://aws.amazon.com/ec2/instance-types/.

[2] Amazon Elastic Compute Cloud Price List. http://aws.amazon.com/de/ec2/pricing/.

[3] Amazon Elastic MapReduce. http://aws.amazon.com/de/elasticmapreduce/ .

[4] Amazon Web Services Dicussion Forum Public Data Sets. https://forums.aws.amazon.com/forum.jspa?forumID=55.

[5] Google App Engine Billing and Budgeting Resources. http://code.google.com/intl/en/appengine/docs/billing.html.

[6] Microsoft Azure Pricing. http://www.microsoft.com/windowsazure/pricing/.

[7] Opennet Rooftop Network. http://www.on-i.de/.

[8] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[9] R. Barrett. Under the Covers of the Google App Engine Datastore. *Google I/O*, 2008.

[10] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. Appscale design and implementation. Technical report, UCSB Technical Report, http://www.cs.ucsb.edu/ ckrintz/papers/appscale2009-02TR.pdf, 2009. Uploaded as online publication to CiteSeer.

[11] E. Ciurana. *Developing with Google App Engine*. Springer, 2009.

[12] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). RFC3626, October 2003. http://www.ietf.org/rfc/rfc3626.txt.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[14] C. Evangelinos and C.N. Hill. Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. *ratio*, 2(2.40):2–34, 2008.

[15] Larry Greenemeier. NSF Teams with Microsoft to Move Scientific Research into the Clouds. *Scientific American*, 2010. http://www.scientificamerican.com/article.cfm?id=nsf-microsoft-cloud.

[16] S. Hazelhurst. Scientific computing using virtual high-performance computing: a case study using the Amazon elastic computing cloud. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, pages 94–103. ACM, 2008.

[17] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman, and P. Maechling. Scientific workflow applications on Amazon EC2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59–66. IEEE, 2010.

[18] N. Leavitt. Inside NOSQL Databases. *Computer*, 43(02), 2010.

[19] N. Leavitt. Will NoSQL Databases Live Up to Their Promise? *Computer*, 43(2):12–14, 2010.

[20] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131. IEEE Computer Society, 2009.

[21] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li. Comparison of Several Cloud Computing Platforms. In *Second International Symposium on Information Science and Engineering*, pages 23–27. Ieee, 2009.

[22] D. Sanderson. *Programming Google app engine*. Oreilly & Associates Inc, 2009.

[23] The DataNucleus project. http://www.datanucleus.org/.

[24] University of Chicago. The Science Clouds Web Site. http://www.scienceclouds.org/.

[25] L. Wang, J. Tao, M. Kunze, A.C. Castellanos, D. Kramer, and W. Karl. Scientific cloud computing: Early definition and experience. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 825–830. IEEE, 2008.

[26] T. White. *Hadoop: The Definitive Guide*. Yahoo Press, 2010.

[27] Maria C. Zacharias. Microsoft and NSF Enable Research in the Cloud. Press Release 10-023, 2010. http://www.nsf.gov/cise/news/2010_microsoft.jsp.