Maximizing Utilization in Private IaaS Clouds with Heterogenous Load through Time Series Forecasting

Tomáš Vondra and Jan Šedivý
Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27
Prague, Czech Republic
vondrto6@fel.cvut.cz, sedivja2@fel.cvut.cz

***Abstract*—This document presents ongoing work on creating a computing system that can run two types of workloads on a private cloud computing cluster, namely web servers and batch computing jobs, in a way that would maximize utilization of the computing infrastructure. To this end, a queue engine called Cloud Gunther has been developed. This application improves upon current practices of running batch computations in the cloud by integrating control of virtual machine provisioning within the job scheduler. For managing web server workloads, we present ScaleGuru, which has been modeled after Amazon Auto Scaler for easier transition from public to private cloud. Both these tools are tested to run over the Eucalyptus cloud system. Further research has been done in the area of Time Series Forecasting, which enables to predict the load of a system based on past observations. Due to the periodic nature of the interactive load, predictions can be made in the horizon of days with reasonable accuracy. Two forecasting models (Holt-Winters exponential smoothing and Box-Jenkins autoregressive) have been studied and evaluated on six server load time series. The autoscaler and queue engine are not yet integrated. Meanwhile, the prediction can be used to decide how many servers to turn off at night or as an internal component for the autoscaling system.**

Keywords - Cloud Computing; Automatic Scaling; Job Scheduling; Real-time Infrastucture; Time Series Forecasting.

I. INTRODUCTION

This paper is an extension of conference article [1].

According to Gartner [2], private cloud computing is currently at the top of the technology hype; but, its popularity is bound to fall due to general disillusionment.

Why? While the theoretical advantages of cloud computing are widely known – private clouds build on the foundations of virtualization technology and add automation, which should result in savings on administration while improving availability. They provide elasticity, which means that an application deployed to the cloud can dynamically change the amount of resources it uses. Another connected term is agility, meaning that the infrastructure can be used for multiple purposes depending on current needs. Lastly, the cloud should provide self-service, so that the customer can provision his infrastructure at will, and pay-per-use, so he will pay exactly for what he consumed.

The problem is that not all of these features are present in current products that are advertised as private clouds. Specifically, this document will deal with the problem of infrastructure agility.

A private cloud can be used for multiple tasks, which all draw resources from a common pool. This heterogenous load can basically be broken down into two parts, interactive processes and batch processes. An example of the first are web applications, which are probably the major way of interactive remote computer use nowadays, the second could be related to scientific computations or, in the corporate world, data mining.

This division was chosen because of different service level measures used in both the fields. While web servers need to be running all the time and have response times in seconds, in batch job scheduling, the task deadlines are generally in units ranging from tens of minutes to days. This allows a much higher amount of flexibility in allocating resources to these kinds of workloads. In other words, while resources for interactive workloads need to always be provisioned in at least the amount required by the offered load, a job scheduler can decide on when and where to run tasks that are in its queue.

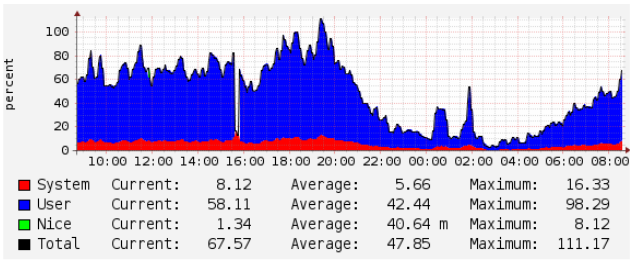

Figure 1. Daily load graph of an e-business website [3]

When building a data center, which of course includes private clouds, the investor will probably want to ensure that it is utilized as much as possible. The private cloud can help achieve that, but not when the entire load is interactive. This is due to the fact that interactive load depends on user activity, which varies throughout the day, as seen in Figure 1.

In our opinion, the only way to increase the utilization of a private cloud is to introduce non-interactive tasks that will fill in the white parts of the graph, i.e., capacity left unused by interactive traffic (which of course needs to have priority over batch jobs).

HPC (High Performance Computing) tasks are traditionally the domain of grid computing. Lately, however, they also began to find their way into the cloud. Examples may be Google's data mining efforts in their private cloud or Amazon's Elastic MapReduce public service [4]. The grid also has the disadvantage that it is only usable for batch and parallel jobs, not interactive use.

Currently, there is not much support for running of batch jobs on private clouds. The well-known scheduling engines Condor [5] and SGE (Sun Grid engine) [6] both claim Amazon EC2 (Elastic Compute Cloud) [7] compatibility, they however cannot control the cloud directly, they only use resources provisioned by other means (See Section II.). (SGE seems to be able to control cloud instances in a commercial fork by Univa, though [8].)

That is why the Cloud Gunther project was started. It is a web application that can run batch parallel and pseudoparallel jobs on the Eucalyptus private cloud [9]. The program does not only run tasks from its queue; it can also manage the VM (virtual machine) instances the tasks are to be run on.

What the application currently lacks is support for advanced queuing schemes (only Priority FCFS (First Come First Served) has been implemented). Further work will include integration of a better queuing discipline, which will be capable of maximizing utilization of the cloud computing cluster by reordering the tasks as to reduce the likelihood of one task waiting for others to complete, while there are unused resources in the cluster, effectively creating a workflow of tasks (see Section V).

The goal is that the scheduler will be fed with data about the likely amount of free resources left on the cluster by interactive processes several hours into the future by a predictor. This will ensure that the cluster is always fully loaded, but the interactive load is never starved for resources.

Prediction of load or any other quantity in time is studied in a branch of statistics called Time Series Analysis and Forecasting. This discipline has also been studied as part of this project and first results are presented in this paper.

This document has five sections. After Section I, Introduction, comes Section II, Related Work, which will present the state of the art in the area of grid schedulers and similar cloud systems. Section III, Cloud Technology, summarizes progress done in cloud research at the Dept. of Cybernetics, mainly on the ScaleGuru autoscaler and the Cloud Gunther job scheduler. Section IV, Time Series, deals with the possibilities for load prediction and evaluates two forecasting methods on server load data. Section V, Future Work, outlines the plans for expansion of the scheduler, mainly to accommodate heterogenous load on the cloud computing cluster. Section VI, Conclusion, ends the paper.

II. Related Work

As already stated, the most notable job control engines in use nowadays are probably SGE [6] and Condor [5]. These were developed for clusters and thus lack the support of dynamic allocation and deallocation of resources in cloud environments.

There are tools that can allocate a complete cluster for these engines, for example StarCluster for SGE [10]. The drawback of this solution is that the management of the cloud is split in two parts – the job scheduler, which manages the instances currently made available to it (in an optimal fashion, due to the experience in the grid computing field), and the tool for provisioning the instances, which is mostly manually controlled.

This is well illustrated in an article on Pandemic Influenza Simulation on Condor [11]. The authors have written a web application, which would provision computing resources from the Amazon cloud and add them to the Condor resource pool. The job scheduler could then run tasks on them. The decision on the number of instances was however left to the users.

A similar approach is used in the SciCumulus workflow management engine, which features adaptive cloud-aware scheduling [12]. The scheduler can react to the dynamic environment of the cloud, in which instances can be randomly terminated or started, but does not regulate their count by itself.

The Cloud Gunther does not have this drawback, as it integrates job scheduling with instance provisioning. This should guarantee that there is no unused time between the provisioning of a compute resource and its utilization by a task, and that the instances are terminated immediately when they are no longer needed.

A direct competitor to Cloud Gunther is Cloud Scheduler [13]. From the website, it seems to be a plug-in for Condor, which can manage VM provisioning for it. Similar to Cloud Gunther, it is fairly new and only features FCFS queuing.

An older project of this sort is Nephele [14], which focuses on real-time transfers of data streams between jobs that form a workflow. It provisions different-sized instances for each phase of the workflow. In this system, the number and type of machines in a job are defined upfront and all instances involved in a step must run at once, so there is little space for optimization in the area of resource availability and utilization.

Aside from cluster-oriented tools, desktop grid systems are also reaching into the area of clouds. For example, the Aneka platform [15] can combine resources from statically allocated servers, unused desktop computers and Amazon Spot instances. It can provision the cloud instances when they are needed to satisfy job deadlines. Certainly, this system seems more mature than Cloud Gunther and has reached commercial availability.

None of these systems deals with the issue of resource availability in private clouds and fully enjoys the benefits of the illusion of infinite supply. To the best of our knowledge, no one has yet dealt with the problem of maximizing utilization of a cloud environment that is not fully dedicated to HPC and where batch jobs would have the status of "filler traffic."

As to time series forecasting, there are efforts to use it on Grids, such as the Network Weather Service (NWS) referenced in a paper by Yang, Foster, and Schopf [16], who describe a better forecasting method for it. The method seems much simpler than the ones being applied in this

article. The NWS project seems to be no longer active, though.

The problems on grids are different from those in clouds. In clouds, we discuss automatic scaling of web servers on identical hardware and data center utilization, whereas in grids, the main problems are prediction of task execution times on heterogenous machines, as described by Iverson, Özgüner, and Potter in [17], and queue wait times and job interarrival times, discussed by Li in [18].

III. CLOUD TECHNOLOGY

A. Eucalyptus

Eucalyptus [9] is the cloud platform that is used for experiments at the Dept. of Cybernetics. It is an open-source implementation of the Amazon EC2 industry standard API (Application Programming Interface) [7]. It started as a research project at the University of California and evolved to a commercial product.

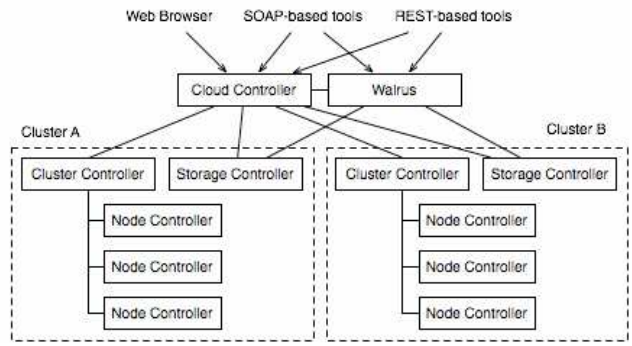

Figure 2. Eucalyptus architecture [9]

It is a distributed system consisting of five components. Those are the Node Controller (NC), which is responsible of running virtual machines from images obtained from the Walrus (Amazon S3 (Simple Storage Service) implementation). Networking for several NCs is managed by a Cluster Controller (CC), and the Cloud Controller (CLC) exports all external APIs and manages the cloud's operations. The last component is the Storage Controller (SC), which exports network volumes, emulating the Amazon EBS (Elastic Block Store) service. The architecture can be seen in Figure 2.

Our Eucalyptus setup consists of a server that hosts the CLC, SC and Walrus components and is dedicated to cloud experiments. The server manages 20 8-core Xeon workstations, which are installed in two labs and 1/4 of their capacity can be used for running VM instances through Eucalyptus NCs. A second server, which is primarily used to provide login and file services to students and is physically closer to the labs, is used to host Eucalyptus CC.

The cloud is used for several research projects at the Cloud Computing Center research group [19]. Those are:

- Automatic deployment to PaaS (Platform as a Service), a web application capable of automatic deployment of popular CMS (Content Management Systems) to PaaS. [20]
- ScaleGuru, an add-on for private clouds, which adds automatic scaling and load balancing support for web applications. [21]
- Cloud Gunther, a web application that manages a queue of batch computational jobs and runs them on Amazon EC2 compatible clouds.

Aside from this installation of Eucalyptus, we also have experience deploying the system in a corporate environment. An evaluation has been carried out in cooperation with the Czech company Centrum. The project validated the possibility of deploying one of their production applications as a machine image and scaling the number of instances of this image depending on current demand. A hardware load-balancer appliance from A10 Networks was used in the experiment and the number of instances was controlled manually as private infrastructure clouds generally lack the autoscaling capabilities of public clouds.

B. ScaleGuru

The removal of this shortcoming is the target of the ScaleGuru project [21], an autoscaling system that can be deployed in a virtual machine in a private IaaS cloud and is able to automatically manage instances of other applications on it.

The software is written in Node.JS with the MongoDB database. It is closely modeled after Amazon Auto Scaling [22], so that users familiar with its structure will easily learn to use ScaleGuru. Therefore, its data model contains Autoscaling Groups, which place lower and upper limits on the number of started instances. Launch Configs then specify the image of the managed application and its parameters. Load Balancers manage the hostnames of the managed services and balanced ports. Lastly, there are Autoscaling Policies and Autoscaling Alarms, which together form the scaling rules such as: "If the CPU Utilization was over 80% for 2 minutes, launch 1 more instance." Using multiple rules, it is possible to create a dynamic response curve.

The program consists of four parts, which are easily replaceable. The Application Core implements the autoscaling logic. It uses the Monitoring component to provide input. Currently it supports collection of CPU utilization, disk and network throughput using an agent on the managed instances. This has the advantage that it is not hypervisor-dependent, but requires the user's cloud API key so that the agent can be injected. If implemented as a service on the private cloud, this is not a problem and has the advantage that the user can sign in to the autoscaler using these keys.

The scaling decisions are implemented through the Cloud Controller component, which supports Amazon EC2 compatible clouds and was tested on Eucalyptus. It can track the state of launched instances and can retry launching on failure. All errors are logged to the web interface. Launched instances are added to Nginx configuration through the Load Balancer Controller.

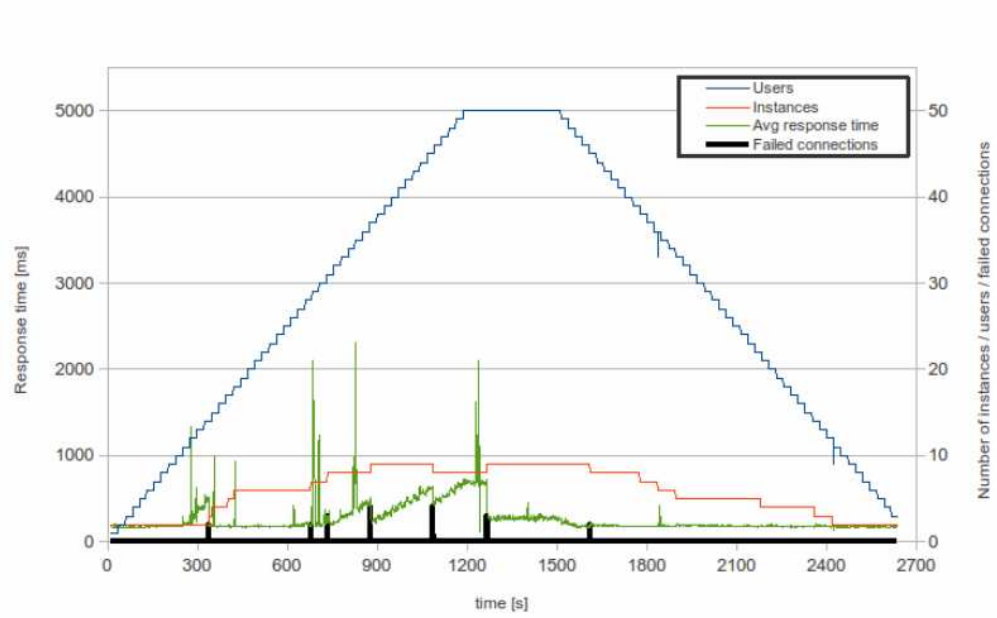

Figure 3. ScaleGuru evaluation [21]

The software was evaluated in a lab setup with Wordpress as the managed application. The PHP version of RUBiS [23], which is a web application created as a benchmarking etalon, was also tried, but it proved to be ill suited for a cloud scaling experiment, as the design of the system is 10 years old and is, contrary to Wordpress, not prepared for horizontal scaling.

A graph from the benchmarking scenario is on Figure 3. In blue is the number of simulated users, who are alternating between thinking (0.5 - 2 s) and waiting for server response. The peak load was about 100 requests per second. In red is the number of instances single CPU and 512 MB RAM on an Intel(R) Core(TM) i3-2100T CPU @ 2.50GHz (2 cores, 4 threads) machine). In green are the response times at the load tester. A drawback of the software load balancer can be seen on the failed connection count (black), which spikes for several hundred milliseconds every time the balancer configuration is reloaded. HAProxy was also tried but had the same problem. The x axis is in milliseconds, y in units of instances and percents of failed connections.

The ScaleGuru application has a modern looking web interface created using Twitter Bootstrap. The monitoring panel, shown on Figure 5, has the number of running instances in green, pending in orange and the red line is average CPU utilization across the autoscaling group. Machine access using a query interface is also possible, it is however currently not Amazon-compatible.

What is important in the context of this paper is that all historical performance data on all autoscaling groups are saved in the database, which enables later analysis using time series methods.

Therefore, the autoscaler will provide input for further experiments on the level of particular applications and will create non-static load in the context of the whole private cloud. A next version of the system could also use the output of the predictor as input for its autoscaling decisions and thus be able to provision capacity for a spike (of a predictable daily or weekly nature), before an actual overload happens.

As far as we know, it is the only piece of autoscaling software, which is installable on a private cloud and fairly universal, and, therefore, suitable for experiments. All other solutions we found were either offered as remotely as Software as a Service or were simple scripts created for a particular project.

C. *Cloud Gunther*

While the ScaleGuru project will also be instrumental for further research, the Cloud Gunther and possibilities for its further development are the main topic of this article.

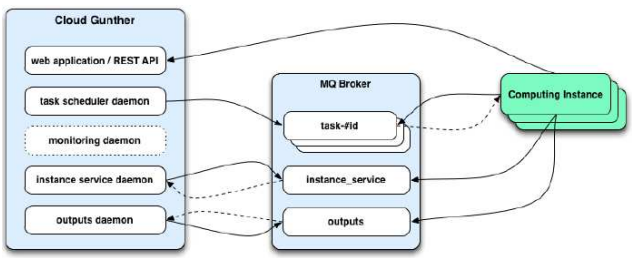

Figure 4. Communication scheme in Cloud Gunther [24]

The application is written in the Ruby on Rails framework and offers both interactive and REST (Representational State Transfer) access. It depends on Apache with mod_passenger, MySQL and RabbitMQ for operation. It can control multiple Amazon EC2 [20] compatible clouds. The queuing logic resides outside the MVC (Model, View, Controller) scheme of Rails, but shares database access with it. The communication scheme is on Figure 4.

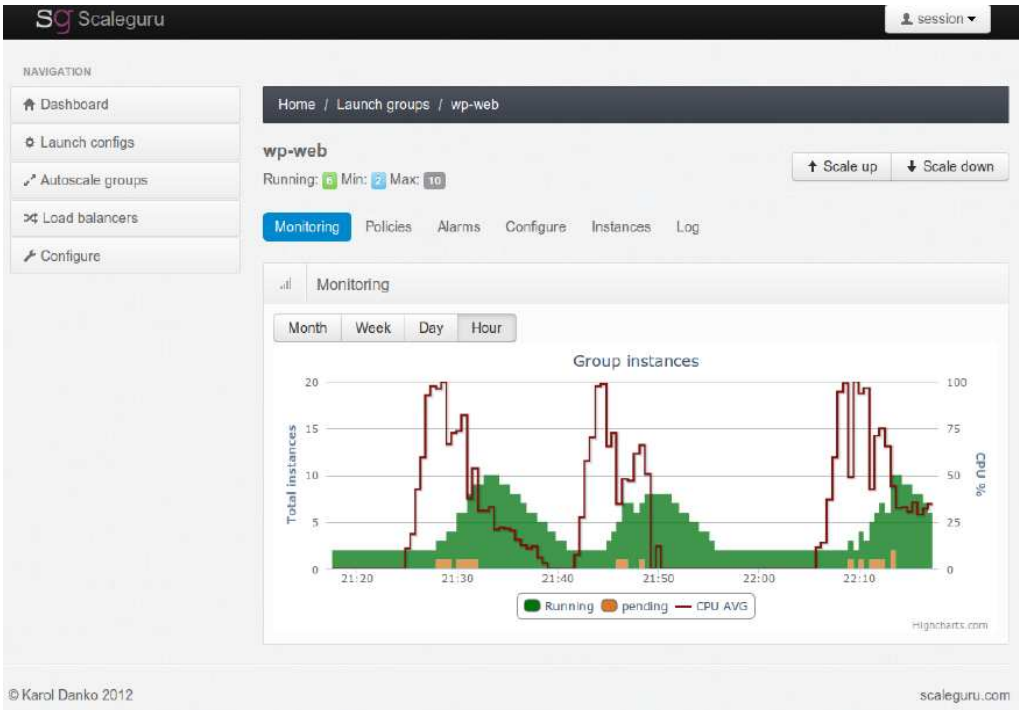

Figure 5. ScaleGuru web interface [21]

The Scheduler daemon contains the Priority FCFS queuing discipline and is responsible for launching instances and submitting their job details to the message broker. The Agent on the instance then retrieves these messages and launches the specified user algorithm with the right parameters. It is capable of running multiple jobs of the same type from the same user, thus saving the overhead of instance setup and teardown.

The two other daemons are responsible for collecting messages from the queue, which are sent by the instances. The Instance Service serves to terminate instances, which have run out of jobs to execute; the Outputs daemon collects standard and error outputs of user programs captured by the launching Agent. A Monitoring daemon is yet to be implemented.

The web application itself fulfills the requirement of multitenancy by providing standard user login capabilities. The users can also be categorized into groups, which have different priorities in the scheduler.

The cloud engine credentials are shared for each cloud (for simpler cloud access via API and instance management via SSH (Secure Shell)).

Each cloud engine has associated images for different tasks, e.g., image for Ruby algorithms, image for Java, etc. The images are available to all users, however when launched, each user will get his own instance.

The users can define their algorithm's requirements, i.e., which image the algorithm runs on and what instance size it needs. There is also support for management of different versions of the same algorithm. They may only differ in command line parameters, or each of them may have a binary program attached to it, which will be uploaded to the instance before execution.

Individual computing tasks are then defined on top of the algorithms. The task consists of input for the algorithm, which is interpolated into its command line with the use of macros, as well as the instance index and total count of instances requested. These values are used by pseudoparallel algorithms to identify the portion of input data to operate on, and by parallel algorithms for directing communication in message passing systems.

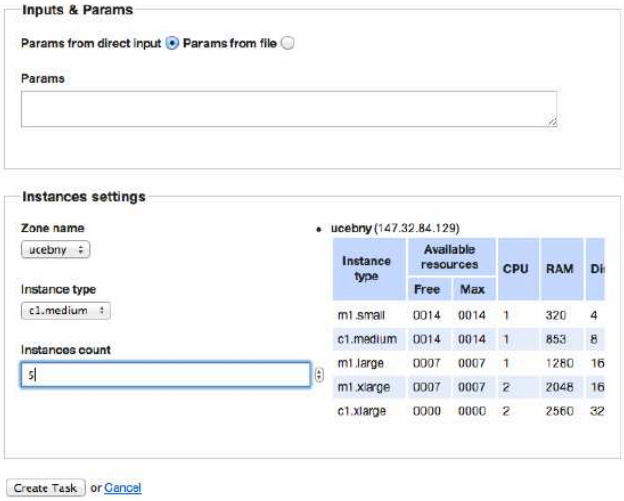

Figure 6. Cloud Gunther – part of the New Task screen [24]

As one can see in Figure 6, the system is ready for private clouds. It can extract the amount of free resources

from Eucalyptus and the scheduler takes it into account when launching new instances.

The Cloud Gunther has been tested on several real workloads from other scientists. Those were production planning optimization, recognition of patterns in images and a multiagent simulation. They represented a parameter sweep workflow, a pseudoparallel task and a parallel task, respectively.

VM images for running the tasks were prepared in cooperation with the users. Usability was verified by having the users set up algorithm descriptions in the web interface. The program then successfully provisioned the desired number of VM instances, executed the algorithms on them, collected the results and terminated the instances.

The main drawback, from our point of view, is that when there are jobs in the queue, the program consumes all resources on the cluster.

This is not a problem in the experimental setting, but in a production environment, which would be primarily used for interactive traffic and would attempt to exploit the agility of cloud infrastructure to run batch jobs as well, this would be unacceptable.

In such a setting, the interactive traffic needs to have absolute priority. For example, if there was a need to increase the number of web servers due to a spike in demand, then in the current state, the capacity would be blocked by Cloud Gunther until some of its tasks finished. It would be possible to terminate them, but that would cause loss of hours of work. A proactive solution to the heterogenous load situation is needed.

IV. TIME SERIES

The sought solution will deal with estimation of the amount of interactive load in time. The interactive traffic needs to have priority over the batch jobs. Therefore, the autoscaler will record the histogram of the number of instances that it is managing. From this histogram, data on daily, weekly and monthly usage patterns of the web servers may be extracted and used to set the amount of free resources for Cloud Gunther.

A similar problem exists in desktop grids. Ramachandran, in article [25], demonstrates the collection of availability data from a cluster of desktop machines and presents a simulation of predictive scheduling using this data. The abstraction of the cloud will shield away the availability of particular machines or their groups, the only measured quantity will be the amount of available VM slots of a certain size.

With a predictor, instead of seeing only the current amount of free resources in the cloud, the batch job scheduler could be able to ask: "May I allocate 10 large instances to a parallel job for the next 4 hours with 80% probability of it not being killed?"

A solution to this question exists in statistics, in a discipline called Time Series Analysis. A good tutorial is written by Keogh [26]. It has very wide coverage, mainly on filtering, similarity measures, Dynamic Time Warping and lower bounds on similarity. However, the solution was found elsewhere, although clustering on particular days and offering the next day after the best match as forecast is also a valid approach and was evaluated as better than the two others presented here in the bachelor thesis of Babka [27] on photovoltaic power plant output prediction.

A. Holt-Winters exponential smoothing

Due to the fact that the ScaleGuru autoscaler was not yet tested in a real environment, it was decided to obtain experimental data from single servers of a web hosting company. These are monitored by Collectd and time series data stored in RRDTool's Round Robin Databases. While examining the documentation for export possibilities, a function by Brutlag [28] was discovered, which uses Holt-Winters exponential smoothing to predict the time series one step ahead and then raise an alarm if the real value is too different from the prediction. This allows to automatically detect spikes in server of network activity.

A good description of exponential smoothing methods including mathematical notation is written by Kalekar [29]. Simple exponential smoothing is similar to moving average. It has a single parameter, α, which controls the weight of the current observation versus the historical value and a single memory that holds the average. It is good for time series that do not exhibit trend or seasonality, and its prediction is a straight line in the mean.

Double or Holt's exponential smoothing takes trend into account. It has 2 parameters, α and β, and a memory of 2, the mean and the slope. The slope is calculated as an exponentially smoothed difference between the current value and predicted mean. Predictions from this model are a straight line from the mean under the average slope.

Lastly, Triple or Holt-Winters exponential smoothing takes seasonality into account. It has 3 parameters, α, β and γ and a memory of 2 plus the number of observations per period. The seasonal memory array holds the factor or addend (depending on whether multiplicative or additive seasonality is used) of each observation point in the season to the exponentially smoothed value, and is itself updated through exponential smoothing. The prediction from this model looks like the average season repeated over in time, starting at the average value and "stair-stepping" with the trend.

Estimation of the parameters can either be done by hand and evaluated using MSE (Mean Squared Error) or MAPE (Mean Average Percentage Error) on the training data (a quick explanation of their significance is in Hyndman [30]), or it can be left to statistics software, which can do fitting by least square error. For the experiments in this paper, the R statistics package [31] was used, particularly the forecast package by Hyndman [32]. The RRDTool implementation is not suitable as it only forecasts one point into the future for spike detection.

An introduction to time series in R, including loading of data, creating time series objects, extracting subsets, performing lags and differences, fitting linear models, and using the zoo library is written by Lundholm [33]. A summary of all available time series functions is in the time series task view [34], while a more mathematical view of the

capabilities including citations of the authors of particular packages is in McLeod, Yu and Mahdi [35].

B. Experiments

1) Loading of data

The evaluation of the method was done on six time series from servers running different kinds of load. The data was first extracted from RRDTool and pushed into MySQL by a bash script, which was being run every day to get data at the desired resolution. The RRD format automatically aggregates data points using maximum, minimum and average, after they overflow the configured age boundaries. Those were (in files created by Collectd) 10 hours in 30 second intervals, 24 h in 60 s, 8 days in 8 minutes, 1 month in 37 min, and 1 year in 7.3 hours.

The chosen initial resolution for experiments was 15 minutes, as the aim is to forecast a) for IaaS clouds, where instance start-up takes about 5 minutes, plus user initialization, and accounting is done in hours, and b) for batch jobs, where the user will probably give task durations in hours or their fractions. Later, it will be evident that this resolution is appropriate for forecasts with the horizon of days, which was the goal of the selection.

The data was then loaded into R (using manual [36]). There was a total of 8159 observations or 2.8 months of data. Time series objects (ts) were created. Their drawback is that observations need to be strictly periodic and the x axis is indexed only by numbers. Any missing values have been interpolated (there was no larger consecutive missing interval). For uneven observation intervals, the "zoo" library may be used, which indexes observations with time stamps [37]. It was not used here, so for clarification: The measurement interval starts with time stamp 1128, which was November 28, and then the count increases every day by 1 irrespective of the calendar as the seasonal frequency was set to 1 day. Therefore, the interval contains Christmas at about 1/3, and it ends on Thursday.

2) Time series diagnostics

The servers included in the experiments have code names oe, bender, lm, real, wn, gaff. In the next paragraph follow their designations and the result of examinations of the time plots of their CPU load time series. This series was also filtered by simple moving average (SMA) with window set to 1 day to obtain deseasonalized trend. The time plots of the series along with best forecasts from both methods are attached in Appendix A.

- oe is a large web shop. It has a clear and predictable daily curve with one weekday higher and weekend and holidays lower (incl. Christmas). Trend is stationary (except Christmas).
- bender is shared PHP webhosting. It has a visible daily curve with occasional spikes. First month shows a decreasing trend, and then it stabilizes.
- lm is a discount server. The low user traffic creates a noisy background load that is dominated by spikes of periodic updates. Trend alternates irregularly between two levels; the duration is on the scale of weeks.
- real is a map overlay service, not much used but CPU intensive (as one map display operation fetches many objects in separate requests). The time plot is a collection of spikes, more frequent during day than night. There are 2 stationary levels, where the first month the load was higher, and then the site was optimized so it went lower.
- wn is PHP hosting of web shops. It has low traffic with a visible daily curve. There is a slow linear additive trend after the first month.
- gaff is a web shop aggregator and search engine. Its daily curve is inverted with users creating background load in the day and a period of high activity due to batch imports during the night. Trend is stationary.

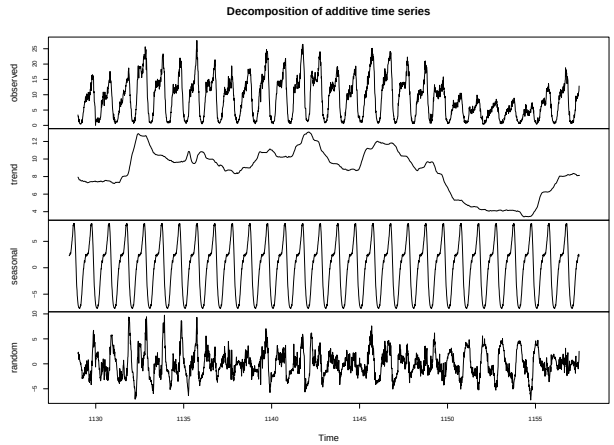

Figure 7. oe series decomposition, from top to bottom: overall time plot, trend, seasonal and random compoment

As suggested in the tutorial by Coghlan [38], which also covers installation of R and packages, as well as Holt-Winters and ARIMA models, the time series were run through seasonal decomposition. For oe, bender and wn, the daily curve was as expected; with gaff, the nightly spike also showed nicely. lm and real surprisingly also show daily seasonality as the spikes are apparently due to periodic jobs. Decomposition of the first month of oe is in Figure 7. We can clearly see the repeated daily curve and a change in trend during Christmas.

Another tool to diagnose time series is the seasonal subseries plot. When applied to the test data, only oe shows clean seasonal behavior. In the bender series, noise may be more dominant than seasonality. The lm series seasonal subseries is also not clearly visible. real clearly shows that traffic on certain hours is higher. For wn, the upward trend is visible in each hourly subseries. gaff shows that the duration of the batch jobs is not always the same so there are large spikes in the morning hours, mainly at the start of the measurement interval. This plot is in Figure 8. It contains 96 subseries because of the 15-min frequency, index 0 is midnight.

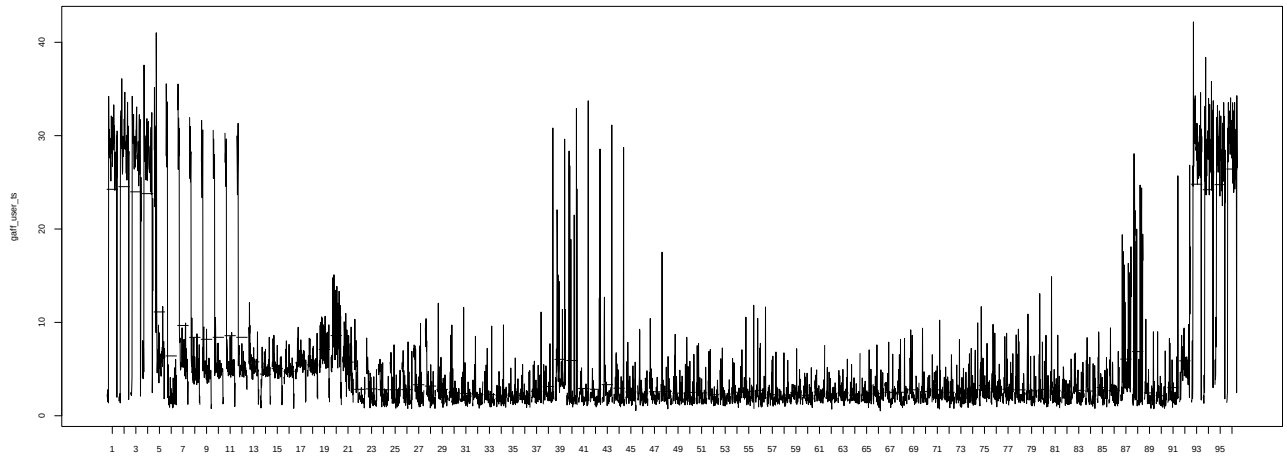

Figure 8. gaff series seasonal subseries plot

3) Model fitting and evaluation

A modified script from Hyndman and Athanasopoulos [39] was used for model fitting and validation. The algorithm first shortens the time series by 3 days at the end and fits a model on it. Then forecasts are created for 6, 24, and 96 hour horizons and compared with the withheld validation data. The result is a table of standard model efficiency measures for each series and interval ("in" meaning in-sample). One more measure was defined in accordance with the goal specified at the beginning of this section – how many validation data points missed the computed 80% prediction intervals in the 3-day forecast (that is 288 points in total).

As to the forecast error measures, the following ones are used: The Mean Error (ME) is a measure of error in absolute scale; it is signed, so it can be used to see a bias in forecasts, but cannot be used for comparison of time series with different scale.

The Root Mean Squared Error (RMSE) measures squared error and is thus more sensitive to outliers. It is best used when the scale of errors is significant. The square root operation returns the dimension to that of the original data.

Mean Absolute Error (MAE) is similar to ME, but ignores the direction of the error by using absolute values.

Mean Percentage Error (MPE) removes the influence of scale from ME by dividing error by the value,

Mean Absolute Percentage Error (MAPE) does the same to MPE. It is probably the best measure for human evaluation.

The Mean Absolute Scaled Error (MASE) is different from the others in that it does not compare the error to the original data, but to the error of the naïve "copy the previous value" forecast method.

For one-step-ahead forecasts, MASE values below one indicate that the evaluated method is better. For larger horizons, this is not true, as the naïve method has more information than the one under evaluation (i.e., always the previous data point). Normally, ME, RMSE, and MAE have the dimension of the original data, MPE and MAPE are in

TABLE I. EVALUATION OF THE HOLT-WINTERS MODEL ON OUT-OF-SAMPLE DATA

	ME	RMSE	MAE	MPE	MAPE	MASE	miss		ME	RMSE	MAE	MPE	MAPE	MASE	miss
oe in	0.003	1.109	0.798	2.776	17.91	1.036		**rea1 in**	-0.03	4.836	3.029	-18.2	38.47	0.398	
oe 6	0.691	1.110	0.829	6.111	7.623	1.076		**rea1 6**	-1.54	7.206	4.878	-68.3	86.06	0.641	
oe 24	0.500	2.461	1.985	-30.4	62.34	2.575		**rea1 24**	-0.28	6.843	4.770	-50.2	71.75	0.627	
oe 96	1.843	4.238	3.223	-26.1	75.49	4.181	2	**rea1 96**	-0.31	7.004	4.916	-56.3	77.77	0.646	84
bend in	-0.06	1.699	1.176	-7.38	23.45	1.110		**rea2 in**	-0.11	7.515	5.973	-68.4	95.76	0.785	
bend 6	0.015	1.280	1.068	-2.11	14.47	1.009		**rea2 6**	-1.78	6.866	5.485	-105	122.1	0.721	
bend 24	-0.36	1.436	1.200	-17.9	27.39	1.133		**rea2 24**	-0.28	8.304	6.619	-88.9	115.6	0.870	
bend 96	-1.33	2.385	1.934	-35.7	41.42	1.826	2	**rea2 96**	-0.30	8.387	6.713	-95.1	122.0	0.883	44
lm1 in	-0.35	5.408	3.832	-10.3	31.63	0.801		**wn in**	-0.01	2.469	1.600	-15.2	43.31	1.047	
lm1 6	3.408	4.839	3.713	18.09	20.46	0.777		**wn 6**	-0.35	1.880	1.553	-11.4	24.44	1.016	
lm1 24	-12.9	17.78	14.81	-119	129.4	3.099		**wn 24**	-1.42	3.617	2.980	-74.8	87.18	1.950	
lm1 96	-27.2	32.23	27.86	-248	251.4	5.830	97	**wn 96**	-1.29	5.151	3.995	-86.4	102.8	2.614	0
lm2 in	0.002	5.638	3.856	-13.5	31.52	0.806		**gaff in**	-0.01	3.562	2.039	-8.90	57.79	1.158	
lm2 6	0.639	5.667	4.625	-6.41	28.14	0.967		**gaff 6**	0.191	7.099	6.449	63.97	465.5	3.663	
lm2 24	-1.04	6.939	5.104	-24.7	40.55	1.068		**gaff 24**	-0.01	6.835	4.308	-8.97	189.3	2.447	
lm2 96	-1.04	7.666	5.624	-29.4	45.83	1.176	14	**gaff 96**	0.622	5.927	4.002	5.364	157.7	2.274	4

percent and MASE is dimensionless. Here, all values are dimensionless as the input data is a time series of CPU load percentages.

The result can be seen in Table I. For lm, two result sets are included. The first is from a triple exponential smoothing model, but as there was a spike at the end of the fitting data, the function predicted an upward trend while the data was in fact stationary. Simple exponential smoothing was then tried, which gave lower error measures and fewer points outside confidence intervals.

A similar problem existed with real. The spikes predicted by the seasonal model missed the actual traffic spikes most of the time. It seems that the series is not seasonal after all, but rather cyclic. The cause for the spikes is random arrivals of requests, as per queuing theory. Cyclicity is discussed in Hyndman [40]. The important outcome is that exponential smoothing models cannot capture it, while autoregressive models can.

The second model for real in the table is double exponential smoothing, which, interestingly, shows higher error measures, but lower number of missed observations. The cause is that the confidence intervals are computed based on the variance of in-sample errors. Therefore, the closer the error magnitude is between in-sample and out-of sample measurement, the more accurate the model is in the "misses" measure.

Automatic model fitting also failed for gaff. The transition from the nightly spike to daily traffic caused the predicted values to be below zero. A manual adjustment of Alpha parameter was necessary. Computed α=0.22, set α=0.69. The problem probably is that the algorithm optimizes in-sample squared error (MSE) and thus it preferred a slower reaction, which mostly missed the spike. The computed trend from this mean was therefore strongly negative. A quicker reaction to the change in mean improved the model, but even then, series with abrupt changes in mean are not good for the Holt-Winters model.

From Table I., we can see that with the Holt-Winters method, some series are predicted well even for the 3 day interval (bender, lm method 2), for some, the forecast is reasonably accurate for the first 6 hour interval and then deteriorates (oe, lm method 1, wn), for others it is inaccurate (real, gaff).

In addition, when the error measures for in-sample data are worse than for out-of-sample, it is a sign of overtraining - the validation data set was closer to "average" than the training data. This is because we were training on a long period including Christmas and verifying on a normal week. Perhaps shortening the training window would be appropriate.

C. Box-Jenkins / ARIMA models

The tutorial [38] suggests using autocorrelation plot on the residuals of the Holt-Winters model. A significant autocorrelation of the residuals means that they have a structure to them and do not follow the character of white noise. All the models showed significant autocorrelation of residuals at both low lags and lags near the period. The Ljung-Box test is a more rigorous proof of randomness of a time series as its null hypothesis is that a group of autocorrelations up to a certain lag is non-significant. It can thus ignore a random spike in the ACF. All the models failed the test in the first few lags.

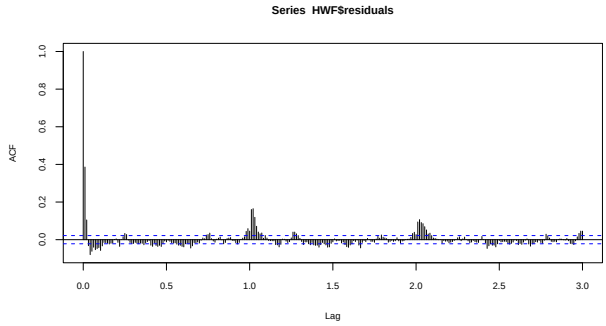

Figure 9. Autocorrelogram of residuals of the H-W model on bender

Having seen autocorrelation plots such as in Figure 9, it was decided to move to better models. ARIMA (Autoregressive, Integrated, Moving average) models are intrinsically based on autocorrelation. They seem to be the state of the art in time series modeling and are a standard in economic prediction (e.g., [39] is a textbook for business schools and MBA).

Neural network methods were also studied, but, as Crone's presentation, which is also a good source on time series decomposition and ARIMA [41], suggests, their forecasting power is equal to ARIMA, only the fitting method is different. It may be more powerful in that it is non-linear and adaptive, but has many degrees of freedom in settings and the result is not interpretable.

As per the NIST Engineering Statistics Handbook [42], chapter 6.4.4.4, which is a good practical source on all methods discussed here, the autoregressive and moving average models were known before, but Box and Jenkins have combined them together and created a methodology for their use.

There are three major steps in the methodology: model selection based on mainly on examination of autocorrelograms (ACF) and partial autocorrelograms (PACF), then model estimation, which uses non-linear least square fitting and/or maximum likelihood and is best left to statistical software, and lastly model validation, which uses ACF and PACF of residuals and the Ljung-Box test.

An autoregressive (AR) model computes the next data point as a linear combination of previous ones, where the number of lagged values considered is determined by the order of the model. The parameters are the mean and the coefficients of each lag. They can be computed by linear least squares fitting. A model of order greater than one with some coefficients negative can exhibit cyclic behavior.

A moving average (MA) model works with errors. The next data point is a linear combination of differences of past lags from the moving average, where the number of lags considered is the order of the model. Again, each term has a parameter that needs to be estimated. The estimation is more difficult as the errors cannot be known before the model

exists, which calls for an iterative non-linear fitting procedure.

The I in ARIMA stands for integrated, which represents the inverse operation to differencing. As the AR and MA models assume that the time series is stationary, meaning that it has stable location and variance, the difference operator can often be used to transform a series to stationary. The model is fitted to the transformed series and an inverse transform is used on the resulting forecast.

Other useful transformations are logarithms and power transforms, which may help if the variance depends on the level. They are both covered by the Box-Cox transform (see [39], chapter 2/4).

D. Experiments

1) Model selection

a) Differencing order

The prerequisite for ARIMA is that the time series is stationary. Manually, stationarity can be detected from the time plot. A stationary time series has constant level and variance, and may not exhibit trend or seasonality. The two last effects should be removed for identification of model order, but are covered by ARIMA models with non-zero differencing order and SARIMA (Seasonal ARIMA), respectively. For series with non-linear trend or multiplicative seasonality, the Box-Cox transform should be used, but that was not the case with the series studied here. Additionally, a non-stationary series will have ACF or PACF plots that do not decay to zero.

The statistical approach to identification of differencing order is through unit root tests (see Nielsen [43]). The root referred to here is the root of the polynomial function of the autoregressive model. If it is near one, any shocks to the function will permanently change the level and thus the resulting series will not be stationary. The standard test for this is Augmented Dickey-Fuller (ADF), which has the null hypothesis of unit root. A reversed test is Kwiatkowski-Phillips-Schmidt-Shin (KPSS), where the null hypothesis is stationarity. There is also a class of seasonal unit root tests that can help specify the differencing order for SARIMA, these are Canova-Hansen (CH) and Osborn-Chui-Smith-Birchenhall (OCSB).

In R, there exist functions ndiffs() and nsdiffs(), which automatically search for the differencing and seasonal differencing order, respectively, by repeatedly using these tests and applying differences until the tests pass (for KPSS and CH), or stop failing (for ADF and OCSB). The default confidence level is 5%. The recommended amount of differencing of the experimental time series obtained from the tests is in Table II on the next page. Columns lm4 and real4 will be explained later.

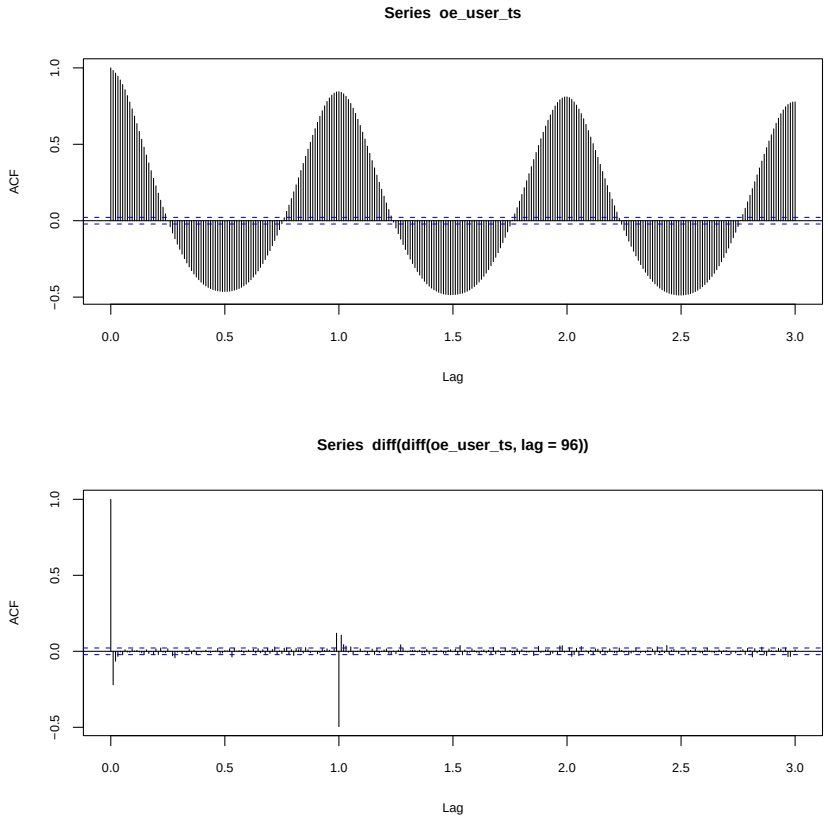

Figure 10. ACF of oe without and with differencing

TABLE II. ORDER OF DIFFERENCING BASED ON UNIT ROOT TESTS

	oe	bender	lm	lm4	real	real4	wn	gaff
ADF	0	0	0	0	0	0	0	0
KPSS	0	1	1	1	1	1	1	0
OCSB	0	0	0	0	0	0	0	0
CH	0	0	0	1	0	1	0	0

It is evident that the ADF and KPSS tests did not agree with each other with the exception of oe and gaff. According to [43], ADF should be considered primary and KPSS confirmatory. The same is said by Stigler in discussion [44], adding that unit root tests have lower sensitivity than KPSS. In the same discussion, Frain says KPSS may be more relevant as a test concretely for stationarity (there may be non-stationary series without a unit root), if we do not assume a unit root based on underlying theory of the time series. It was also used by Hyndman in the auto.arima() function for iterative model identification.

According to manual heuristic approaches, such as presented by Nau [45], an order of seasonal differencing should always be used if there is a visible seasonal pattern. It also suggests applying a first difference if the ACF does not decay to zero. An example of the impact of first and seasonal differencing on stationarity and thus legibility of an ACF plot is in Figure 10.

The ACF and PACF functions on the test data were looked at with and without differencing with the result that differencing rapidly increases the decay of the ACF function on all series except real.

Moreover, from the ACF of lm and real, it seems there is a strong periodicity of 4 hours. These two series will be also tested with models of this seasonal frequency and will be denoted as lm4 and real4, as in Table II.

For the purpose of order identification, seasonal and then first differences have been taken. It was decided to test if the models fitted with this order of differencing, following the heuristic approach, are better or worse than those with differencing order identified by statistical tests.

b) Order identification

Identification of model order was done using heuristic techniques from [39], [42], [45], and [46]. After seasonal and first differencing is applied in the necessary amount to make the time series look stationary to the naked eye, so that its autocorrelograms converge to zero, the ACF and PACF functions are looked at. The number of the last lag from the beginning where PACF is significant specifies the maximum reasonable order of the AR term, similarly the last significant lag on ACF specifies the MA order. The order of the seasonal autoregressive and moving average terms is obtained likewise, but looking at lags that are multiplies of the seasonal period.

The observed last significant lags and resulting maximum model orders are summed in Table III. Model parameters are denoted as ARIMA(p, d, q)(P, D, Q), where p is the order of the AR term, d is the amount of differencing and q is the order of the MA term. The second parenthesis specifies the seasonal model orders.

TABLE III. LAST SIGNIFICANT LAGS AND MODEL ORDERS

	PACF	ACF	seas. PACF	seas. ACF	estimated maximal model parameters
oe	5	3	11	1	ARIMA(5,1,3)(11,1,1)
bender	17	4	9	1	ARIMA(17,1,4)(9,1,1)
lm	15	16	8	1	ARIMA(15,1,16)(8,1,1)
lm4	9	2	11	∞	ARIMA(9,1,2)(11,1,0)
real	1	2	11	1	ARIMA(1,0,2)(11,1,1)
real4	13	2	11	1	ARIMA(13,1,2)(11,1,1)
wn	39	3	10	1	ARIMA(39,1,3)(10,1,1)
gaff	18	2	6	1	ARIMA(18,1,2)(6,1,1)

Looking at the two variants of lm, the expectation is that the first will perform better, as the non-seasonal part covers the second period of 4 hours. This is not true for real vs. real4.

2) Model estimation

When trying to fit models with high seasonal order, a limitation of the ARIMA implementation in R was found. The maximal supported lag is 350, which with a period of 96 (24 hours * 4 observation per hour) means that the seasonal lag is limited to 3.

Furthermore, the memory requirements of seasonal ARIMA seem to be exponential with the number of data points. A machine with 1 GB of RAM could not handle the 2.8 months of data with lag 288. This constraint is not documented. The experiment had to move to a machine with 32 GB RAM, where computing a model with seasonal order 3 took 7.6 GB RAM, more on subsequent runs as R is a garbage collected language.

For the course of this experiment, the order of the seasonal components will be limited to three, as it should be sufficient when forecasting for a horizon of about a day. The alternatives, which will be examined in further experiments, are to reduce the resolution to 1 hour, which will enable lags up to 12 days.

A model of this sort was fitted on oe, and it did not lead to a better expression of the weekly curve (at least not by visual inspection). With this resolution, it will be however possible to use a seasonal period of one week, which should be able to capture the day-to-day fluctuations. Similarly, we would reduce resolution is we were trying to capture monthly or yearly seasonality.

Another approach, suggested by Hyndman [47], is to model the seasonality using a Fourier series and to use non-seasonal ARIMA on the residuals of that model. This should enable fitting on arbitrarily long seasonal data. This may lead to overfitting, though, as the character of the time series is subject to change over longer time periods.

For the actual parameter estimation, the Arima() function with the model order as parameter can be used. There is however a way to automate a part of the identification-estimation-validation cycle and that is the auto.arima() function. This function repeatedly fits models with different parameters and then returns the one that has minimal Akaike Information Criterion (AIC). This criterion prefers models with lower likelihood function and contains a penalization for the number of degrees of freedom of the model; therefore, it should select the model that best fits the data,

but not variations of the same model with superfluous parameters.

The auto.arima() function has two modes depending on the "stepwise" parameter (see help(auto.arima) in R). With this set to TRUE, it does a greedy local search, which selects the best model from previous step and examines its neighborhood in the state space given by adding or subtracting one to each parameter. It continues, until no model in the neighborhood has lower AIC.

The second mode searches from ARIMA(0,0,0)(0,0,0) upwards and based on the description, it should search until the ceiling set for each parameter. The actual behavior however seems to be that is stops when the last iteration examined did not bring any gain. Both search modes are thus prone to getting stuck in a local minimum.

To better specify the models, the auto.arima() function was used on each time series with three sets of parameters. In the first run, it was started from zero with stepwise=FALSE and with ceilings set to the parameters estimated in Table III. In the second run, stepwise was set to TRUE and the ceilings were left at the pre-estimated parameters plus one to account for differencing; the starting values were set to be the same as the ceilings, as, theoretically, the parameters in Table III should be the maximal meaningful numbers, but a model with lower orders might be better. This was tested in the third run, where the starting values remained and the ceilings were effectively removed.

The same procedure was then repeated with the differencing orders computed by OCSB and KPSS. As it is difficult to identify model parameters by naked eye without differencing, the same initial parameters have been used. Please note that the AIC values of models with unequal differencing order are not comparable, while goodness-of-fit test results and prediction errors are.

3) Model validation

As already discussed in Subsection IV/C "Box-Jenkins models", the validation entails manual examination of the autocorrelation plot of residuals and use of the Ljung-Box goodness-of-fit (GOF) test. Table IV contains the models that resulted from the three runs of auto.arima() as described, along with their AIC values, the lag of the first significant autocorrelation and the lag after which the Ljung-Box test failed. Left side is for models with differencing order set to one, right side has differencing set by unit root tests.

The outcome from Table IV is, that is cannot be conclusively said whether it is better to always use seasonal differencing or not. Of the six time series, three have the best fitting model in the left half of the table and three in the right half. However, it seems that in the cases where the non-differenced models were better, the gain in the goodness-of-fit functions was lower than the other way round. It is also interesting that in two of the three cases (oe and gaff), the difference is not only in seasonal, but also in first differencing. It may be a good idea to follow the recommendation of the KPSS test, but always use seasonal differencing, but there is not enough data to say it with certainty.

A more solid fact is that all the best models come from the third row of the table. Of the three tried here, the best algorithm for model selection is to use auto.arima() in greedy mode, starting with parameters identified from ACF and PACF, and leave it room to adjust the parameters upwards.

E. Comparison of the two model families

The last part of the experiment entailed computing forecasts based on the fitted ARIMA models and comparing them with out-of-sample data. The same validation algorithm

TABLE IV. PARAMETERS OF THE ESTIMATED ARIMA MODELS AND THEIR VALIDATION MEASURES

	model	AIC	sig. ACF	fail. GOF		model	AIC	sig. ACF	fail. GOF
oe	ARIMA(0,1,2)(1,1,2)	23016.97	12	14	**oe**	ARIMA(4,0,3)(2,0,2)	23231.26	22	23
	ARIMA(6,1,1)(1,1,2)	23109.12	12	15		ARIMA(5,0,4)(3,0,2)	23259.8	21	27
	ARIMA(5,1,3)(2,1,3)	23066.43	16	20		*ARIMA(5,0,3)(3,0,3)*	*23220.86*	*21*	*27*
bend	ARIMA(1,1,1)(2,1,1)	28082.19	4	6	**bend**	ARIMA(1,1,1)(3,0,2)	27989.07	22	6
	ARIMA(17,1,5)(3,1,2)	27580.45	52	129		ARIMA(17,1,4)(3,0,2)	27812.25	26	60
	ARIMA(17,1,3)(3,1,3)	*27504.15*	*58*	*172*		ARIMA(14,1,1)(3,0,3)	27801.06	17	58
lm	ARIMA(1,1,1)(2,1,1)	47569.93	5	5	**lm**	ARIMA(1,1,3)(1,0,2)	48517.21	5	4
	ARIMA(16,1,17)(2,1,2)	47210.57	94	144		ARIMA(15,1,17)(3,0,1)	48155.89	43	144
	ARIMA(17,1,17)(2,1,3)	*47195.88*	*98*	*500+*		ARIMA(15,1,18)(3,0,1)	48152.6	70	144
lm4	ARIMA(2,1,1)(1,1,1)	49151.84	5	5	**lm4**	ARIMA(1,1,3)(0,0,2)	50789.93	4	4
	ARIMA(10,1,3)(12,1,1)	48398.45	11	14		ARIMA(10,1,2)(12,0,2)	48570.04	10	28
	ARIMA(11,1,2)(16,1,5)	48138.57	21	30		ARIMA(12,1,2)(15,0,4)	48342.89	21	30
real	ARIMA(1,1,1)(2,1,1)	47872.32	4	3	**real**	ARIMA(0,1,2)(3,0,0)	48873.62	4	4
	ARIMA(2,1,3)(2,1,2)	47800.56	1	1		ARIMA(2,1,3)(3,0,2)	48732.74	1	1
	ARIMA(6,1,8)(3,1,3)	46972.94	6	6		ARIMA(10,1,12)(3,0,3)	47344.03	8	9
rea4	ARIMA(2,1,1)(1,1,1)	47574.67	3	3	**rea4**	ARIMA(1,1,1)(3,0,0)	48897.42	3	3
	ARIMA(1,1,3)(1,1,2)	47612.58	2	2		*ARIMA(12,1,2)(11,0,1)*	*47438.97*	*21*	*24*
	ARIMA(4,1,5)(1,1,7)	47373.03	8	7		*ARIMA(12,1,2)(11,0,1)*	*47438.97*	*21*	*24*
wn	ARIMA(4,1,2)(2,1,2)	35599.79	9	14	**wn**	ARIMA(2,1,3) with drift	36214.64	10	9
	ARIMA(40,1,2)(2,1,2)	35608.54	55	500+		ARIMA(39,1,4)(1,0,2)	36177.56	59	95
	ARIMA(39,1,1)(2,1,3)	*35596.67*	*55*	*500+*		ARIMA(38,1,5)(1,0,3)	36146.1	64	191
gaff	ARIMA(2,1,3)(0,1,2)	21501.92	5	5	**gaff**	ARIMA(3,0,0)(1,0,1)	21847.8	5	5
	ARIMA(19,1,3)(0,1,2)	21387.26	42	52		*ARIMA(18,0,3)(1,0,2)*	*21717.96*	*43*	*88*
	ARIMA(17,1,4)(0,1,3)	21118.64	42	88		*ARIMA(18,0,3)(1,0,2)*	*21717.96*	*43*	*88*

TABLE V. EVALUATION OF THE ARIMA MODELS ON OUT-OF-SAMPLE DATA

	MAPE in	MAPE 6	MAPE 24	MAPE 96	miss		MAPE in	MAPE 6	MAPE 24	MAPE 96	miss
oe	13.43	*7.12*	75.27	94.99	8	**oe**	13.40	8.13	37.52	53.16	22
	13.39	7.13	77.74	98.23	*7*		13.22	8.71	33.96	48.88	22
					failed		*13.16*	8.85	*31.41*	*46.35*	24
bend	19.32	*14.56*	22.18	22.21	25	**bend**	*18.28*	15.34	45.21	41.32	*13*
	18.51	15.51	*20.58*	*21.3*	42		18.79	21.42	36.53	34.38	87
					failed						failed
lm	24.93	19.14	*19.70*	*22.34*	17	**lm**	25.49	17.23	19.80	23.22	19
	23.94	14.79	20.10	25.15	20		23.98	15.74	21.01	26.91	21
					failed		23.97	15.98	21.11	27.02	21
lm4	25.50	13.22	23.36	28.93	8	**lm4**	28.77	26.94	44.30	51.17	*7*
	24.73	*11.66*	22.19	30.21	24		24.61	12.93	21.81	29.02	21
	24.16	15.57	20.29	23.45	19		24.47	12.51	19.88	25.73	21
real	36.26	85.66	73.58	80.20	85	**real**	38.18	72.49	62.91	64.49	59
	37.13	88.91	76.86	83.95	94		38.18	87.33	74.95	81.07	81
					failed		37.56	69.18	52.74	58.08	*39*
rea4	37.67	58.08	48.30	54.23	59	**rea4**	40.83	53.41	47.86	70.73	43
	37.99	53.44	43.22	*45.75*	40		*36.10*	*50.40*	*41.59*	46.06	49
	37.03	55.33	43.54	46.34	61		*36.10*	*50.40*	*41.59*	46.06	49
wn					failed	**wn**	42.03	*24.30*	78.80	82.33	102
	37.68	36.26	*51.12*	*50.25*	*59*		38.92	24.36	64.94	71.68	79
					failed		38.96	26.33	64.16	70.09	78
gaff	*37.62*	*160.67*	128.23	112.77	61	**gaff**	37.65	170.08	136.91	124.57	*59*
	38.19	165.29	*125.89*	*109.42*	60		38.26	165.48	133.85	119.44	*59*
	38.56	187.57	129.88	110.55	61		38.26	165.48	133.85	119.44	*59*

was used as in the case of Holt-Winters models, to facilitate model comparison. The result is in Table V. To conserve space, only MAPE (Mean Average Percentage Error) is shown. The four columns are for in-sample error and forecast errors in horizons 6, 24, and 96 hours. The ordering of models is the same as in Table IV.

Fitting of the forecasts was something of a disappointment, as all of the models with seasonal differencing (the left half of Table IV) that were selected as best using the GOF measures have failed to produce forecasts. The cause was likely the seasonal MA part of the model that was one or two orders higher that the originally identified ceiling. That resulted in an overspecified model where the MA polynomial was not invertible. Invertibility is a prerequisite for the computation of variances of the parameters [48], which in turn are needed to compute confidence intervals for a prediction. Hence, these models were fitted and had a likelihood function and in-sample errors, but could not be used for forecasts with confidence bounds.

When fitting ARIMA models in R, one needs to carefully observe the output for warnings such as:

```
In sqrt(z[[2]] * object$sigma2) : NaNs produced
```

for least-squares fitting, or for maximum likelihood:

```
Error in optim(init[mask], armafn, method =
optim.method, hessian = TRUE, :
non-finite finite-difference value [1]
In log(s2) : NaNs produced
```

because then the prediction will produce wrong results or fail:

```
In predict.Arima(object, n.ahead = h) :
MA part of model is not invertible
```

Therefore, if using auto.arima() beyond the ceiling identified from ACF and PACF, there is a high risk of the model failing and thus it may not be a good idea for automatic forecasts. If that happens, lowering the order or the seasonal MA or MA part should help.

As to the selection of the best model for forecasts, the selection based on out-of sample forecast errors (mainly looking at the 24 and 96-hour horizons) corresponds to the one based on goodness-of-fit criteria. In the case where the model fails to produce forecasts, the next-best one based on GOF can be selected. The second row (ceilings from ACF and PACF adjusted downward by auto.arima()) produced the best result, except on oe and lm, where, however, the difference is seems to be small.

As whether to always use seasonal differencing, the experiment is inconclusive. In the case of oe, there was a significant gain in accuracy by not using it, in the case of wn and bender, the opposite is true.

Looking at the "misses" criterion, one could say that Holt-Winters is better. However, that outcome might be skewed. The criterion counts the number of data points that missed the 80% confidence bounds in the 3-day forecast. That time period contains a total of 288 points, 20% of that is 57.6, and that is the count of data points that are by definition allowed to miss the bounds.

Therefore, the result of this comparison is that the confidence bounds on ARIMA are more accurate, or at least tighter than on Holt-Winters. If this method is to be used as proposed by this article, the confidence level used has to be adjusted upwards to 95 or 99%, depending on the overload sensitivity of the computer infrastructure.

Comparing the two model families using the MAPE error measure, the outcome is that ARIMA did produce better forecasts than Holt-Winters, except for the 6-hour forecasts on oe and bender, and also that simple exponential smoothing outperformed both seasonal methods on 24 and 96-hour forecasts on lm.

V. FUTURE WORK

Future work planned on the Cloud Gunther can be split into two categories. First and more important is the consideration of interactive load also present on the cluster, which will require a rewrite of the queue engine to utilize the output of the predictor. Second is integration of better queuing disciplines to bring it up to par with existing cluster management tools. Two ideas for that are presented in Subsections A and B. Section C discusses the problem of resource sharing on modern computers.

A. Out-of-order scheduling

Using load predictions to maximize load of course assumes a scheduler that will be capable of using this information. Our vision is a queue discipline that internally constructs a workflow out of disparate tasks. The tasks, each with an associated estimate of duration, will be reordered so that the utilization of the cloud is maximized.

For example, when there is a job currently running on 20 out of 40 slots and should finish in 2 hours, and there is a 40 slot job in the queue, it should try to run several smaller 2 hour jobs to fill the free space, but not longer, since that would delay the large job.

These requirements almost exactly match the definition of the Multiprocessor scheduling problem (see [49]). Since this is a NP-hard class problem, solving it for the whole queue would be costly. The most feasible solution seems to come from the world of out-of-order microprocessor architectures, which re-order instructions to fully utilize all execution units, but only do so with the first several instructions of the program. The batch job scheduler will be likewise able to calculate the exact solution with the first several jobs in the queue, which will otherwise remain Priority FCFS.

B. Dynamic priorities

The estimation of job duration is a problem all for itself. At first, the estimate could be done by the user. Later, a system of dynamic priorities could be built on top of that.

The priorities would act at the level of users, penalizing them for wrong estimates, or better, suspending allocation of resources to users whose tasks have been running for longer time than the scheduler thought.

Inspiration for this idea is taken from the description of the Multilevel Feedback Queue scheduler used historically in Linux [50]. However, the scheduler will set priorities for users, not processes, and allocate VMs to tasks, not jiffies to threads. It also will not have to be real-time and preemptive, making the design simpler.

The scheduler's estimate of process run time could be based on the user estimates, but also on the previous run time of processes from the same task or generally those submitted by the same user for the same environment. That would lead to another machine learning problem.

C. Resource sharing

When we actually have both kinds of traffic competing for resources of the cloud, resource-sharing problems may affect the performance of the system and raise the observed requirements of the interactive traffic.

The effects of different kinds of algorithms on their surroundings will have to be benchmarked and evaluated. We may have to include disk and network bandwidth requirements in the model of the batch job and decide, which jobs may and which may not be run in a shared infrastructure, or ensure their separation through bandwidth limiting.

Second, even if we set up the private cloud so that CPU cores and operating memory are not shared, we still have to count with the problem of shared cache memory. This has been researched by several groups. Gusev and Ristov [51] benchmarked this by running multiple instances of a linear equation solver in a virtualized environment, and Babka, et.al., [52] measured the problem when concurrently running several benchmark kernels from SPEC2000.

VI. CONCLUSION

The cloud presents a platform that can join two worlds that were previously separate – web servers and HPC grids. The public cloud, which offers the illusion of infinite supply of computing resources, will accommodate all the average user's needs, however, new resource allocation problems arise in the resource-constrained space of private clouds.

We have experience using private cloud computing clusters both for running web services and batch scientific computations. The challenge now is to join these two into a unified platform.

The ScaleGuru autoscaling system offers an opportunity to get hand-on experience with automatic scaling, as most other systems are either very simple or are being developed in the commercial sector without public access to source codes.

The Cloud Gunther, although not ready for commercial deployment, already has some state of the art features, like the automatic management of cloud computing instances and a REST-compliant web interface. It also differs from other similar tools by its orientation towards private cloud computing clusters.

In the future, it could become a unique system for managing batch computations in a cloud environment primarily used for web serving, thus allowing to exploit the dynamic nature of private cloud infrastructure and to raise its overall utilization.

This article also presented two methods of time series forecasting, used otherwise mainly in economic forecasts, and which could be applied to server load data. These methods were tested on six time series of CPU load, some of which are web servers with a well defined daily curve (oe, bender, wn), and some have a load of more unpredictable nature (lm, real, gaff).

As it is expected that the cloud will contain mostly load-balanced web servers as the variable component, we think that these methods are viable for further research in the optimization of cloud computing.

ACKNOWLEDGMENTS

Credit for the implementation of Cloud Gunther, mainly the user friendly and cleanly written web application goes to Josef Šín. The ScaleGuru application and all its modules were written by Karol Danko.

We thank the company Centrum for providing hardware for our experiments and insights on private clouds from the business perspective.

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant no. SGS13/141/OHK3/2T/13, Application of artificial intelligence methods to cloud computing problems.

REFERENCES

[1] T. Vondra and J. Šedivý, "Maximizing Utilization in Private IaaS Clouds with Heterogenous Load," in CLOUD COMPUTING 2012: The Third International Conference on Cloud Computing, GRIDs, and Virtualization, IARIA, 22 July 2012, pp. 169-173.

[2] D. M. Smith, "Hype Cycle for Cloud Computing," Gartner, 27 July 2011, G00214915.

[3] T. Vondra and J. Šedivý, "Od hostingu ke cloudu," Research Report GL 229/11, CTU, Faculty of Electrical Engineering, Gerstner Laboratory, Prague, 2011, ISSN 1213-3000.

[4] R. Grossman and Y. Gu, "Data mining using high performance data clouds: experimental studies using sector and sphere," in Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '08). ACM, New York, NY, USA, 2008, pp. 920-927, doi: 10.1145/1401890.1402000.

[5] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," in 8th International Conference on Distributed Computing Systems, 1988, pp. 104-111.

[6] W. Gentzsch, "Sun Grid Engine: towards creating a compute power grid," in Proceedings of the first IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001, pp. 35-36.

[7] "Amazon Elastic Compute Cloud (EC2) Documentation," Amazon, <http://aws.amazon.com/documentation/ec2/> 27 May 2012.

[8] T. P. Morgan, "Univa skyhooks grids to clouds: Cloud control freak meets Grid Engine," The Register, 3rd June 2011, <http://www.theregister.co.uk/2011/06/03/univa_grid_engine_cloud/ > 19 March 2012.

[9] "Installing Eucalyptus 2.0," Eucalyptus, <http://open.eucalyptus.com/wiki/EucalyptusInstallation_v2.0> 19 March 2012.

[10] "StarCluster," Massachusetts Institute of Technology, < http://web.mit.edu/star/cluster/index.html> 11 May 2012.

[11] H. Eriksson, et al., "A Cloud-Based Simulation Architecture for Pandemic Influenza Simulation," in AMIA Annu Symp Proc., 2011, pp. 364–373.

[12] D. de Oliveira, E. Ogasawara, K. Ocaña, F. Baião, and M. Mattoso, "An adaptive parallel execution strategy for cloud-based scientific workflows," Concurrency Computat.: Pract. Exper. (2011), doi: 10.1002/cpe.1880.

[13] "Cloud Scheduler," University of Victoria, <http://cloudscheduler.org/> 11 May 2012.

[14] D. Warneke and O. Kao, "Nephele: efficient parallel data processing in the cloud," in MTAGS '09: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, November 2009, doi: 10.1145/1646468.1646476.

[15] R. N. Calheiros, C. Vecchiola, D. Karunamoorthya, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds," Future Generation Computer Systems 28 (2012), pp. 861-870, doi: 10.1016/j.future.2011.07.005.

[16] L.Yang, I. Foster, and J.M. Schopf, „Homeostatic and Tendency-based CPU Load Predictions," in Proceedings of IPDPS 2003, April 2002, p.9.

[17] M. Iverson, F. Özgüner, and L.C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," in Proceedings of Eighth Heterogeneous Computing Workshop, HCW'99, IEEE, 1999, pp. 99-111.

[18] H. Li, „Performance evaluation in grid computing: A modeling and prediction perspective," in CCGRID, Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007, IEEE, pp. 869-874.

[19] J. Šedivý, "3C: Cloud Computing Center," CTU, Faculty of Electrical Engineering, dept. of Cybernetics, Prague, <https://sites.google.com/a/3c.felk.cvut.cz/cloud-computing-center-preview/> 19 March 2012.

[20] T. Vondra, P. Michalička, and J. Šedivý, „UpCF: Automatic deployment of PHP applications to Cloud Foundry PaaS," 2012, unpublished.

[21] K. Danko, "Automatic Scaling in Private IaaS," Master's Thesis, CTU, Faculty of Electrical Engineering, Supervisor T. Vondra, Prague, 3 January 2013.

[22] "Amazon Auto Scaling Documentation," Amazon, <http://aws.amazon.com/documentation/autoscaling/> 12 March 2013

[23] ObjectWeb Consortium, "RUBiS: Rice University Bidding System," 2003, <http://rubis.ow2.org/> 12 March 2013.

[24] J. Šín, "Production Control Optimization in SaaS," Master's Thesis, CTU, Faculty of Electrical Engineering and University in Stavanger, Department of Electrical and Computer Engineering, Supervisors J. Šedivý and C. Rong, Prague, 20 December 2011.

[25] K. Ramachandran, H. Lutfiyya, and M. Perry, "Decentralized approach to resource availability prediction using group availability in a P2P desktop grid," Future Generation Computer Systems 28 (2012), pp. 854–860, doi: 10.1109/CCGRID.2010.54.

[26] E. Keogh, "A Decade of Progress in Indexing and Mining Large Time Series Databases," in Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 14 September 2006.

[27] M. Babka, "Photovoltaic power plant output prediciton," Bachelor's Thesis, CTU, Faculty of Electrical Engineering, Supervisor P. Kordík, Prague, 1 May 2011.

[28] J. Brutlag, "Aberrant behavior detection in time series for network monitoring," in Proceedings of the 14th USENIX conference on System administration, 2000, pp. 139-146.

[29] P.S. Kalekar, "Time series forecasting using Holt-Winters exponential smoothing," Kanwal Rekhi School of Information Technology, 6 December 2004.

[30] R.J. Hyndman, "Hyndsight - Forecast estimation, evaluation and transformation," 10 November 2010 <http://robjhyndman.com/hyndsight/forecastmse/> 12 March 2013.

[31] R Development Core Team "R: A language and environment for statistical computing," R Foundation for Statistical Computing,Vienna, Austria, ISBN 3-900051-07-0 (2010), <http://www.R-project.org> 12 March 2013.

[32] R.J. Hyndman, "forecast: Forecasting functions for time series," R package version 4.0, 2011, <http://CRAN.R-project.org/package=forecast> 12 March 2013.

[33] M. Lundholm, "Introduction to R's time series facilities," ver. 1.3, 22 September 2011, <http://people.su.se/~lundh/reproduce/introduction_ts.pdf> 12 March 2013.

[34] R.J. Hyndman, „CRAN Task View: Time Series Analysis," 10 March 2013, <http://cran.r-project.org/web/views/TimeSeries.html> 12 March 2013.

[35] A.I. McLeod, H. Yu, and E. Mahdi, „Time Series Analysis in R," Handbook of Statistics, Volume 30, Elsevier, 27 July 2011, <http://www.stats.uwo.ca/faculty/aim/tsar/tsar.pdf> 12 March 2013.

[36] R Development Core Team "R Data Import/Export." R Foundation for Statistical Computing,Vienna, Austria, ver. 2.15.3, 1 March 2013, ISBN 3-900051-10-0, <http://www.R-project.org> 12 March 2013.

[37] G. Grothendieck, "Time series in half hourly intervals- how do i do it?" R-SIG-Finance news group, 27 Septemper 2010, < https://stat.ethz.ch/pipermail/r-sig-finance/2010q3/006729.html> 12 March 2013.

[38] A. Coghlan, "Little Book of R for Time Series," 2010, <http://a-little-book-of-r-for-time-series.readthedocs.org/en/latest/> 12 March 2013.

[39] R.J. Hyndman and G. Athanasopoulos, "Forecasting: principles and practice, chapter 8/9 Seasonal ARIMA models," online textbook, March 2012, <http://otexts.com/fpp/8/9/> 12 March 2013.

[40] R.J. Hyndman, "Cyclic and seasonal time series," in Hyndsight, 14 December 2011, <http://robjhyndman.com/hyndsight/cyclicts/> 12 March 2013.

[41] S.F. Crone, "Forecasting with Artificial Neural Networks," Tutorial at the 2005 IEEE Summer School in Computational Intelligence EVIC'05, Santiago, Chile, 15 December 2005, <http://www.neural-forecasting.com/tutorials.htm> 12 March 2013.

[42] NIST/SEMATECH, „Chapter 6.4. Introduction to Time Series Analysis," in e-Handbook of Statistical Methods, created 1 June 2003, updated 1 April 2012, <http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4.htm> 12 March 2013.

[43] H.B.Nielsen, „Non-Stationary Time Series and Unit Root Tests," Lecture for Econometrics II, Department of Economics, University of Copenhagen, 2005, <http://www.econ.ku.dk/metrics/Econometrics2_05_II/Slides/08_unitroottests_2pp.pdf> 12 March 2013.

[44] K. Bhattach, M. Stigler, and J.C. Frain, "Which one is better?" Discussion in RMetrics, 1 Janueary 2010, <http://r.789695.n4.nabble.com/Which-one-is-better-td991742.html> 12 March 2013.

[45] R.F. Nau, „Seasonal ARIMA models," Course notes for Decision 411 Forecasting, Fuqua School of Business, Duke University, 16 May 2005, <http://people.duke.edu/~rnau/seasarim.htm> 12 March 2013.

[46] „Chapter 4: Seasonal Models," in STAT 510 - Applied Time Series Analysis, online course at Department of Statistics, Eberly College of Science, Pennsylvania State University, 2013, < https://onlinecourses.science.psu.edu/stat510/?q=book/export/html/50 > 12 March 2013.

[47] R.J. Hyndman, "Forecasting with long seasonal periods," in Hyndsight, 29 September 2010, <http://robjhyndman.com/hyndsight/longseasonality/> 12 March 2013.

[48] R Development Core Team "arima0: ARIMA Modelling of Time Series – Preliminary Version" in R-Documentation, R Foundation for Statistical Computing,Vienna, Austria, 2010, ISBN 3-900051-07-0, <http://stat.ethz.ch/R-manual/R-patched/library/stats/html/arima0.html> 12 March 2013.

[49] "Multiprocessor scheduling," in Wikipedia: the free encyclopedia, San Francisco (CA): Wikimedia Foundation, 12 March 2012, <http://en.wikipedia.org/wiki/Multiprocessor_scheduling> 19 March 2012.

[50] T. Groves, J. Knockel, and E. Schulte, "BFS vs. CFS - Scheduler Comparison," 11 December 2011 < http://slimjim.cs.unm.edu/~eschulte/data/bfs-v-cfs_groves-knockel-schulte.pdf > 11 May 2012.

[51] M. Gusev and S. Ristov, "The Optimal Resource Allocation Among Virtual Machines in Cloud Computing," in CLOUD COMPUTING 2012: The Third International Conference on Cloud Computing, GRIDs, and Virtualization, IARIA, 22 July 2012, pp. 36-42.

[52] V. Babka, P. Libič, T. Martinec, and P.Tůma, "On The Accuracy of Cache Sharing Models," in Proceedings of ICPE 2012, Boston, USA, ACM, April 2012, pp. 21-32, ISBN 978-1-4503-1202-8.

APPENDIX A – EXPERIMENTAL TIME SERIES AND THEIR FORECASTS FROM HOLT-WINTERS AND BOX-JENKINS

The next page contains the forecasts of each examined time series from the best model of exponential smoothing and ARIMA methods.

The exponential smoothing on in the left half of the page, ARIMA on the right. The series are, from top to bottom: oe, bender, lm, real, wn, gaff.

The graphs contain the last week of the time series to present their character. The blue line then represents the point forecasts; the orange area is the 80% confidence band and the yellow area the 95% confidence band. Overlaid as "o" symbols are the actual data points, which were recorded during the forecast horizon.

It is not important to read the axes of the graphs, the scale is 2 days per tick on the x axis and in percent of an unspecified CPU on y. The character of the time plot and the response of the forecasting algorithms is important.